VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webové rozhraní pro správu procesů výkonnostního testování SIP infrastruktury

Web Interface for Process Management of SIP Infrastructure Benchmarking

2013 Pavol Noha

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Pavol Noha**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Webové rozhraní pro správu procesů výkonnostního testování SIP infrastruktury
Web Interface for Process Management of SIP Infrastructure Benchmarking

Zásady pro vypracování:

Cílem diplomové práce je vývoj webové služby pro správu a management aplikací sloužících pro výkonnostní testování SIP infrastruktury. Algoritmy testování vychází z návrhu RFC pro SIP benchmarking [1] doplněného o další nové myšlenky [2], které byly implementovány ve formě několika oddělených aplikací v linuxovém prostředí, podrobnější popis je uveden v literatuře [2-4]. Diplomant rozebere dostupná řešení a na základě získaných znalostí navrhne a vytvoří rozhraní umožňující spuštění jednotlivých procesů nutných pro provedení výkonnostního testu, jejich kontrolu a zpracování dostupných výsledků poskytovaných těmito procesy v pravidelných intervalech. Veškeré testy budou ukládány do databáze, ze které půjde získat informace o parametrech spuštěného testu a jeho výsledcích. Výsledkem práce bude softwarový balík pro OS linux zjednodušující a zpřehledňující práci s nástrojem SIPp a pracující nad webovým serverem dle výběru diplomanta.

1. Dostupná řešení pro webovou správu procesů a aplikací pod OS Linux.
2. Srovnání webových serverů umožňujících webovou správu procesů.
3. Návrh a realizace rozhraní pro výkonnostní testování SIP infrastruktury.
4. Praktické nasazení a otestování navrženého řešení.

Seznam doporučené odborné literatury:

[1] Davids,C., Gurbani,V., Poretsky,S., Methodology for Benchmarking SIP Networking Devices, RFC Draft, last change - March 2011
[2] Voznak,M., Rozhon, J. Approach to stress tests in SIP environment based on marginal analysis, In Journal SPRINGER: Telecommunication Systems, 11p., June 2011, ISSN 1018-4864 (Print), ISSN 1572-9451 (Online), DOI: 10.1007/s11235-011-9525-1
[3] Voznak,M., Rezac,F., Tomala, K. SIP Penetration Test System, Proceedings The 34th Conference on Telecommunications and Signal Processing, pp. 504-508, August 2010, Baden near Vienna, Austria, ISBN 978-963-88981-0-4
[4] Voznak,M., Rozhon,J. SIP Back to Back User Benchmarking, Proceedings The Sixth International Conference on Wireless and Mobile Communications, pp. 92-96, Published by IEEE Computer Society, September 2010, University of Valencia , Spain, ISBN 978-0-7695-4182-2, DOI 10.1109/ICWMC.2010.86

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Ing. Miroslav Vozňák, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 1. května 2013 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. května 2013 .....

Rád bych na tomto místě poděkoval panu doc. Ing. Miroslavu Vozňákovi Ph.D. za cenné připomínky a odborné rady, kterými přispěl k vypracování této diplomové práce. Dále děkuji panu Ing. Janu Rozhonovi za poskytnuté konzultace k implementaci rozhraní. Rád bych také poděkoval mé rodině za pochopení a podporu, kterou mi v době mého studia projevovali.

Abstrakt

Cílem diplomové práce je vytvořit webové rozhraní pro správu procesů výkonnostního testování SIP infrastruktury programem SIPp. První část práce pojednává o protokolu SIP a jeho testovacím nástroji SIPp. Náplň druhé části je analýza, návrh řešení a následná implementace rozhraní.

Klíčová slova: SIP, SIPp, Testovací rozhraní, Správa procesů

Abstract

The main goal of thesis is creation of web interface for process management of SIP infrastructure Benchmarking by Sipp tool. The first part is focused on the knowledge about the SIP protocol and its test tool SIPp. The subject of the second part is analysis and design of solution followed by implementation of interface.

Keywords: SIP, SIPp, Test interface, Process Management

Seznam použitých zkratek a symbolů

SIP	–	Session Initiation Protocol
DAO	–	Data Access Object
JPA	–	Java Persistence Api
GNU	–	General Public License
DDoS	–	Distributed Denial of Service
VoIP	–	Voice over IP
CLI	–	Command Line Interface
UAC	–	User Aagent Client
UAS	–	User Agent Server
MGCP	–	Media Gateway Control Protocol
UDP	–	User Datagram Protocol
IETF	–	Internet Engineering Task Force
HTTP	–	HyperText Transfer Protocol
IPv6	–	Internet Protocol version 6
JDK	–	Java Development Kit
API	–	Application Programming Interface)
RMI	–	Remote Method Invocation
JDBC	–	Java DataBase Connectivity
POJO	–	Plain Old Java Object
EJB	–	Enterprise Java Beans
ORM	–	Object Relational Mapping
JDO	–	Java Data Objects
JMS	–	Java Messaging Service
MVC	–	Model View Controller
AOP	–	Aspect Oriented Programming
JAR	–	Java application ARchive
WAR	–	Web application ARchive
TCP	–	Transmission Control Protocol
JSP	–	Java Server Pages
CSS	–	Cascading Style Sheets
HTML	–	HyperText Markup Language
PID	–	Process IDentifier

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Žijeme v době, kdy verbální komunikace už dávno není doménou analogových telefonů, nýbrž přístrojů digitálních. Lidé používají různé komunikační nástroje jako Skype, VoIP, aby se mohli vzájemně spojit a řešit různé peripetie, problémy pracovního charakteru, popřípadě si plánovat volnočasové aktivity či si jen povídat. Práce má za úkol přiblížit VoIP [2] a jeho protokol SIP. Jedná se o službu, která umožnuje telefonování prostřednictvím internetové sítě. V praxi to znamená, že klient dostane veřejné telefonní číslo, z něhož se dovolá na kterékoli místo na světě a to jak na levné linky, tak i do mobilních sítí. Na toto veřejné telefonní číslo se klientovi také dovolá kdokoli. Klient tedy nepozná rozdíl oproti stávající telefonní lince. S rozšířením SIP protokolů nastala potřeba testovat chování sítě při různých registracích účastníku a samozřejmě u hovoru jako takového. Pro tyhle účely byl vyvinut nástroj SIPp, který poskytuje CLI interface pro spouštení jak klientských UAC procesů, tak serverových UAS procesů VoIP sítě. Ačkoliv SIPp tester je zajisté velice silná aplikace na testováni SIP protokolu chybu v ní přeci jen najdeme. A to právě z hlediska usability. Proto jsem se s pomocí svého vedoucího diplomové práce rozhodl naprogramovat webové rozhraní, které je schopné obsluhovat jednotlivé SIP procesy, čili interaguje s CLI příkazi SIPp-u v bashi a ukladá výsledky do průhledných tabulek a grafů bez toho, aby uživatel musel znát strukturu příkazů, které jsou mnohdy dosti komplikované.

Teoretická část práce popisuje jak SIP protokol, tak SIPp tester a dává uživateli základní přehled, k čemu je protokol a jeho tester určen. Dále popisuje technologie, které byly pro vývoj a nasazení rozhraní použité.

Praktická část naproti tomu popisuje, jak probíhal vývojový proces od analýzy až po implementaci systému. Tato část je zakončená přehlednou uživatelskou dokumentací systému.

2 SIP vs SIPp

Smyslem této kapitoly je přiblížit uživateli jak protokol, tak tester a zároveň ukázat, jak jednoduché je pracovat s danou technologií.

2.1 SIP jako protokol

V současnosti existuje několik signalizačních protokolů. Kromě pár firemních řešení jsou používány protokoly MGCP (Media Gateway Control Protocol), H.323 a SIP (Session Initiation Protocol) [4]. Dvěmi hlavními standardizovanými protokoly na scéně signalizace, jsou protokoly H.323 a SIP.

Protokol H.323 vznikl a je podporován organizací ITU-T, a proto je svým provedením bližší telefonním normám. Jde o poměrně složité a strukturu celé rodiny protokolů řešící jednotlivé signalizační úlohy. Protokoly používají binární zápis, což ztěžuje proces ladění. Poměrně složitá je rovněž rozšiřitelnost. Teprve ve třetí verzi byla implementována podpora transportního protokolu UDP. Donedávna byl protokol H.323 běžnější než protokol SIP, ale situace se zřejmě obrací.

Protokol SIP, viz [1] vznikl pod křídly organizace IETF (Internet Engineering Task Force). Jde o textový protokol, který je strukturou podobný například poštovnímu protokolu SMTP (Simple Mail Transfer Protocol) nebo protokolu HTTP (HyperText Transport Protocol). Tělo zprávy je tvořeno textovými položkami ve formě <název>:<hodnota>, které popisují předávané informace. Textová podstata protokolu napomáhá nejen jednoduchému ladění, ale především snadné šiřitelnosti.

Protokol SIP je typu klient-server, tudíž komunikace probíhá výměnou dvou typů zpráv, požadavků a odpovědí. Klient i server jsou logickou součástí jednoho prvku. Pokud je v textu nadále zmiňován termín klient, je tím míněn telefon nebo softwarová aplikace u uživatele. Pojmem server jsou označovány aplikační servery služby, které poskytují klientům další služby jako registrace, lokalizace, zpoplatňování atd. Protokol SIP lze použít i pro další účely, například síťové hry, instant messaging a další.

```
INVITE sip:mamut@iptel.org SIP/2.0.
Max-Forwards: 10.
Record-Route: <sip:195.113.222.3;ftag=5DAA94E7;lr=on>.
Via: SIP/2.0/UDP 195.113.222.3;branch=z9hG4bK0a5d.90580ee2.0.
Via: SIP/2.0/UDP 195.113.134.233:5062;branch=z9hG4bK2E1FD348.
CSeq: 262 INVITE.
To: <sip:mamut@iptel.org>.
Proxy-Authorization: Digest username="bbb", realm="ces.net", nonce="43788
    e90381194d66364fced4dc7097828391e81", uri="sip:mamut@iptel.org", cnonce="abcdefghi",
    nc=00000001, response="ed4adec8Content-Type: application/sdp.
From: "Franta Vomacka" <sip:bbb@ces.net>;tag=5DAA94E7.
Call-ID: 379332994@195.113.134.233.
Subject: sip:bbb@ces.net.
Content-Length: 234.
User-Agent: kphone/4.2.
Contact: "Franta Vomacka" <sip:bbb@195.113.134.233:5062;transport=udp>.
```

```
P-hint: outbound.
Remote-Party-ID: "Franta Vomacka" <sip:950070101@ces.net>;party=calling;id-type=
    subscriber;privacy=off; screen=yes.
.
v=0.
o=username 0 0 IN IP4 195.113.134.233.
s=The Funky Flow.
c=IN IP4 195.113.134.233.
t=0 0.
m=audio 33728 RTP/AVP 0 97 8 3.
a=rtpmap:0 PCMU/8000.
a=rtpmap:3 GSM/8000.
a=rtpmap:8 PCMA/8000.
a=rtpmap:97 iLBC/8000.
a=fmtp:97 mode=30.
```

Výpis 1: Příklad SIP zprávy INVITE

2.1.1 Typy požadavků

První položka požadavku obsahuje jeho typ, jež slouží jako označení klienta nebo serveru, kterému je požadavek adresován (Request-URI) a také verzi protokolu. Důležité jsou následující požadavky:

- **INVITE:** Metoda INVITE slouží k přizvání uživatele nebo služby k dané relaci. Tělo zprávy obsahuje popis relace (spojení).
- **ACK:** Metoda ACK potvrzuje, že klient v pořádku přijal odpověď na INVITE dotaz.
- **BYE:** Klient používá metodu BYE k oznámení protistraně, že hodlá ukončit hovor. Metoda BYE může být vyslána jak volaným, tak volajícím.
- **CANCEL:** Metoda CANCEL ukončuje nevyřízený požadavek se stejnou identifikací, tedy položkami Call-ID, To, From a pořadovým číslem požadavku Cseq. Vyřízené požadavky již metoda CANCEL neovlivní (požadavek je vyřízen, pokud byla odeslána konečná odpověď, např. 200 OK).
- **REGISTER:** Metoda REGISTER je používána k registraci současné adresy klienta u SIP serveru, který ji předá lokalizační službě.

2.1.2 Typy odpovědí

Odpovědi se dělí do uvedených kategorií podle kódu odpovědi na:

- **1xx:** Prozatimní odpovědi jako požadavek přijat, vyzvání
- **2xx:** Úspěch. Požadavek je přijat, pochopen a akceptován
- **3xx:** Přesměrování. Je třeba vytvořit nově upravený požadavek

- **4xx:** Chyba klienta. Špatná syntaxe požadavku, požadavek nemůže být proveden
- **5xx:** Chyba serveru. Server není schopen provést platný požadavek
- **6xx:** Globální chyba. Požadavek nelze provést na žádném serveru

Odpovědi s kódem 200 a výše jsou konečné, jejich přijetí uzavírá transakci.

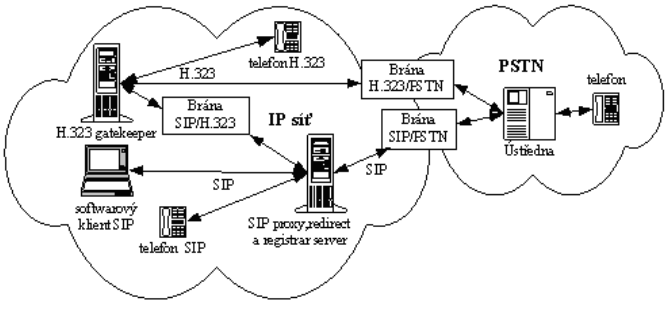
2.1.3 Důležité hlavičky SIP

Níže uvedené položky jsou podstatným výběrem z mnoha možných hlaviček:

- **To:** Adresa volaného
- **From:** Adresa volajícího klienta
- **Via:** Adresa klienta, který vysílá požadavek či adresa serverů, přes něž požadavek prošel, a kudy se bude vracet i odpověď
- **Call-Id:** Unikátní identifikace volání
- **Contact:** Skutečná a aktuální adresa klienta
- **Record-Route:** Seznam adres serverů, které chtějí dostávat veškerou komunikaci náležející k hovoru
- **Route:** Posloupnost adres serverů, přes něž je požadavek směrován na klienta, ke kterému má tento požadavek dorazit.
- **Request-URI:** Aktuální adresát požadavku. Údaj se vyskytuje v první řádce požadavku za metodou (typem požadavku)

2.1.4 Pojmy a prvky architektury služby SIP

Následující odstavec popisuje základní pojmy a prvky, se kterými se setkáme v prostředí SIP protokolu.



Obrázek 1: SIP architektúra

- **SIP adresa:** Uživatel služeb SIP je identifikován pomocí adresy SIP, jinak také SIP URI (Uniform Resource Identifikator). SIP adresa funguje na stejném principu jako adresa e-mail a má nejčastěji následující formát: „sip:user@host“. Je obvyklé, aby až na předponu sip:, byla shodná s e-mail adresou. Částí user může být kromě jména také telefonní číslo, pak ale mívá adresa spíše předponu tel:. Část host je buď doménové jméno nebo adresa IP. Adresa může obsahovat množství dalších nepovinných parametrů jako číslo portu. Pokud není uvedeno, předpokládá se použití všeobecně známého portu 5060.

- **Transakce:** Soubor požadavku a k němu příslušejících odpovědí. Například BYE a 200 OK.

- **User Agent:** Sip prvky se mohou skládat ze dvou logických částí: User Agent Client (UAC) a User Agent Server (UAS). Logická část, která generuje dotaz, vystupuje po dobu transakce jako UAC. A UAS je logická část, která vytváří odpovědi. I tato role platí jen po dobu transakce.

- **SIP Server:** SIP server je obvykle odpovědný za uživatele v doméně. Pracuje v proxy nebo redirect módu. V redirect módu sdělí informace o poloze volaného, získané lokalizační službou zpět volajícímu a ten navazuje spojení přímo na novou adresu. V proxy módu pokračuje navazování spojení prostřednictvím serveru.

- **Location service:** Lokalizační služba je služba poskytující informace o současné poloze uživatele, tedy o IP adrese. Služba může používat například Radius protokol nebo databázový stroj s tabulkou kontaktů.

- **Registrar:** Registrační server zpracovává požadavek REGISTER a informace o poloze klienta předává lokalizační službě, která obsluhuje příslušnou oblast.

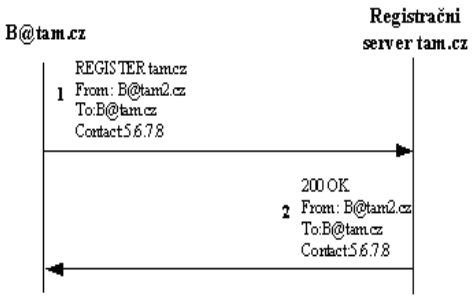
- **Transaction stateful proxy:** Tato proxy udržuje stav transakce, tj. od přijetí požadavku až do odeslání konečné odpovědi.

- **Call statefull proxy:** Tato proxy udržuje stav dialogu od prvního požadavku INVITE až po ukončovací BYE požadavek.

SIP server, lokalizační služba a registrační server bývají často implementovány dohromady.

2.1.5 Registrace

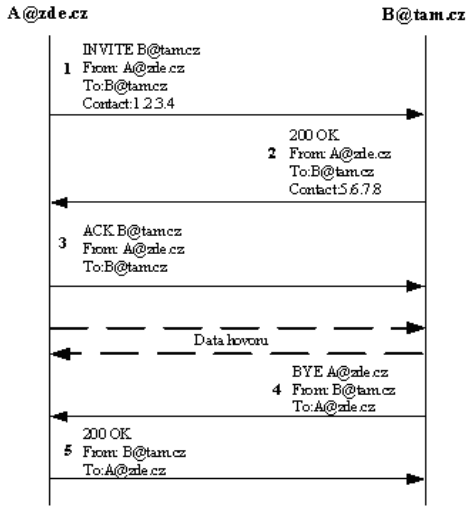
Úspěšný registrační proces je tvořen požadavkem REGISTER a odpovědí 200 OK. Adresa SIP uvedená v položce To s příslušnou aktuální adresou v položce Contact vytvoří v registračním serveru časově omezený záznam, který je předán lokalizační službě. Tato informace umožňuje dosažení uživatele nezávisle na poloze použitého klienta.



Obrázek 2: SIP registrace

2.1.6 Navázání a ukončení spojení

Spojení je navazováno pomocí procedury, která je označována jako „three-way handshake“. Klient A, jenž chce zavolat klientu B, vyšle požadavek INVITE (1). obsahující popis nabízeného spojení. Dorazila-li zpráva v pořádku a byla pochopena, pošle klient B prozatímní odpověď, kterou můžeme chápat jako vyzváněcí tón. Pokud se klient B rozhodne hovor přijmout, vysílá zpět odpověď OK (2) s návratovým kódem 200. Odpověď OK obsahuje kodeky, adresy IP a čísla portů, na kterých bude klient B očekávat data od protistrany. Závěrečnou fází je zaslání zprávy ACK (3) klientem A protistraně, která potvrzuje přijetí odpovědi klienta B. Proces navazování spojení je kompletní a může začít vlastní rozhovor. Pokud chce některý z účastníků hovor ukončit, pošle protistraně požadavek BYE (4), který bude potvrzen odpovědí 200 OK (5).



Obrázek 3: SIP hovor

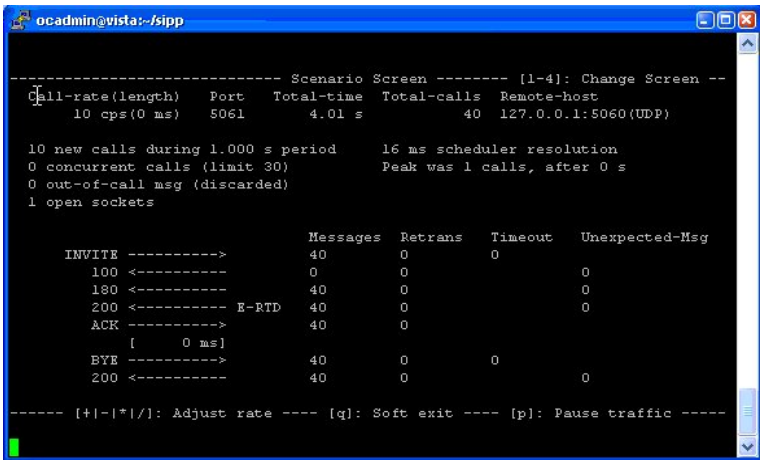
Při popisu protokolu SIP byla použita data z webu sdružení Cesnet [9].

2.2 SIPp jako tester

SIPp je volný Open Source testovací nástroj a generátor hovorů pro SIP protokol. Obsahuje pár základních scénařů (UAS a UAC), vytváří a spouští hovory za pomoci INVITE a BYE požadavků. Taktéž dokáže číst XML scénaře od velmi jednoduchých po velice komplexní procesy daného hovoru. SIPp dále umí dynamicky zobrazovat statistiky o právě bežících testech(rating hovorů, statistiky zpráv) a generovat tyto statistiky do CSV souborů.

Mezi další výhody patří podpora IPv6, TLS, SCTP, SIP autentifikace, podmíněné scénaře, UDP přeposílaní, odolnost vůči chybám (timeout volání, obrana protokolu), speciální proměnné hovoru, možnost připojit atributy z externího csv souboru k imitaci živých uživatelů a nastavitelnost akcí(logování, ukončení hovoru) po obdržení zprávy. SIPp dokáže posílat také média(RTP) v hovoru a to přehráním buď RTP echo nebo RTP pcap. Zatím co byl SIPp optimalizovaný pro volání a zátežové testy, může být použit ke spuštení jednoduchého hovoru, přičemž je uživatel obeznámem s výsledkem hovoru(prošel/selhal).

SIPp může být prakticky použit k testování různých SIP proxy, SIP média serverů, SIP/x brán, SIP PBX ...Je také velice užitečný k simulaci stovek uživatelů volajících náš SIP systém [5] - [8].



Obrázek 4: SIPp test

2.2.1 Instalace

Instalace SIPpu je velice jednoduchá. Pokud používáme linux distribuci, je možno jej najít v repozitáři pod názvem *sip-tester*, nebo stáhnout přímo z oficiálnich stránek produktu. Kvůli nutnosti instalace balíku OpenSSL ve verzi $\geq$ 0.9.8 doporučuji stáhnout přímo a zkompilovat. Balík OpenSSL slouží k autentizaci uživatelů při volání či registraci do sítě.

```
# tar -xvzf sipp-xxx.tar.gz
# cd sipp
# autoreconf -ivf
# ./configure --with-openssl
# make
```

Výpis 2: Instalace SIPp-u s podporou OpenSSL

Další balíky na kterých je SIPp závislý:

- **libc6** ($\geq$ 2.5-5) GNU C knihovna
- **libgcc1** ($\geq$ 1:4.2-20070516) GCC podpůrná knihovna
- **libncurses5** ($\geq$ 5.4-5) Sdílené knihovny pro manipulaci s terminálem
- **libpcap0.7** Systémové rozhraní pro odchyt paketů na uživatelské úrovni
- **libstdc++6** GNU standartní C++ knihovna

SIPp běží také na systému Windows XP a novějších verzích, a to z důvodu podpory protokolu IPv6.

2.2.2 Příklad použití

Mezi dvě základní formy, jak SIPp tester použít, je spuštění již vzpomínaných UAC čili klientských nebo UAS serverových testů.

```
# ./sipp -sf uac.xml -inf pbx_uac.csv -m 1 -nd -i 192.168.10.1 -p 5060 127.0.0.1:5061 -fd 1
# ./sipp -sf uas.xml -m 1 -nd -i 192.168.10.2 -p 5060 -fd 1
```

Výpis 3: Příklad spuštení SIPp UAC a UAS testů

3 SIPp webové rozhraní

S nástupem a popularizací cloudových technologií se přirozeně už při zrodu myšlenky na vytvoření grafické nádstavby pro technologii SIPp tlačila ven idea, že by rozhraní mělo být realizováno formou webové aplikace. Samozřejmě vzhledem k faktu, že je služba licencovaná pod GNU licencí, veškeré využití je bez poplatků. Cílem je poskytnout uživatelům protokolu SIP bezplatnout bránu k nástrojům na otestováni zvolené SIP infrastruktury bez nutnosti instalace jakýchkoliv nástrojů. Uživatel má možnost požádat vlastníka/admina dané větvě o přidání do seznamu testerů a ten pak může využívat všech druhů testů a funkcionalit aplikace. V následujících sekcích Vás obeznámím s kladnými i zápornými stránkami rozhraní.

3.1 Výhody rozhraní

Aplikace beží na severu, což znamená, že se uživatel nemusí starat o:

- Scénaře - Jak UAC tak UAS jsou uploadované uživatelem přímo na server
- Výsledky testů - Všechny výsledky jsou uložené v DB, místo v externích CSV
- Připojené CSV soubory s externí uživatelskou konfigurací - uploadované na server
- Instalaci a konfiguraci aplikace
- Ochranu dat - je použita velice důkladná autentizace
- Použití externích nástrojů na generováni grafů a tabulek z hodnot výsledků testů - aplikace je generuje sama.

Další výhodou je, že webové rozhraní je schopno pracovat s příkazy SIPpu multivláknově čili správce aplikace může v externím konfiguračním souboru nastavit počet vláken odpovídajících počtu procesorových jader, se kterými může aplikace pracovat. Což znamená, že čím vykonnější server máme k dispozici, tím náročnější testy můžeme provádět.

3.2 Nevýhody rozhraní

Nevýhody aplikace jsou minimální, avšak i webové rozhraní má své nedostatky:

- Nemožnost registrace nových účastníků - nemožnost registrace je způsobena především faktem, že testovací aplikace může být využita v nesprávných rukou jako silný nástroj k napadnutí sítě DDoS útokem, viz [3]. Jako jedna z možností se jevila myšlenka na omezení počtu generovaných hovorů uživatelem, ale tento zásah by připravil uživatele o možnost provádět regulérní zátěžové testy. To znamená, že oprávnění k užívaní aplikace dostanou jen lidé, které si admin přidá do listu testerů.

- Důvěra v externí autority - vlastník nebo správce rozhraní se může rozhodnout dále nepokračovat v poskytovaní dané služby, čímž přichází uživatel o svá testovací data a scénaře. V další verzi bude tento krok ošetřen modulem pro export všech důležitých dat.

4 Správa procesů

4.1 Řešení pro webovou správu procesů pod OS Linux

Správa procesů pod OS Linux může být řešena buďto na úrovni programovacího jazyka nebo jako bash skript. Jelikož jsem si k vypracování vybral jazyk Java, který podporuje multithreading, každý proces je zastoupen objektem starajícím se o chod procesu a implementaci metod pro jeho správu.Pro naši aplikaci je klíčovým faktorem zajistit spuštění několika procesů testeru sipp, nejlépe aby byly odstínené od uživatele(background mod) z java kódu. Java implementuje k tomuhle účelu abstraktní třídu *Process*. Metody *ProcessBuilder.start()* a *Runtime.exec* vytvářejí přirozený proces a navrací instanci třídy *Process*, která může být použita k ovládání daného procesu nebo k získavání dat o procesu. Třída *Process* dále poskytuje funkce k uskutečnění vstupu procesu, výstupu procesu, čekání na dokončení procesu, kontrolu, zda-li je proces stále aktivní a ukončení procesu. Třídu *Process* pak používá objekt implementující rozhraní *Callable*, které je velice podobné *Runnable*, ale na rozdíl od něj dokáže vracet výsledek volaného vlákna, v našem případě sipp procesu. Výsledek představuje PID procesu, na jehož základě můžeme vykonávat další operace na systémové úrovni, a proces jako takový je reprezentován svým unikátním identifikátorem. Závisle od platformy a jazyka, existuje několik různých implementací objektu pro správu procesů v OS Linux.

Naproti tomu je v bashi správa procesů doménou skriptů. Bash podporuje proměnné, řídící struktury, cykly a logické výrazy a dokáže velice snadno volat systémové příkazy.

4.2 Webové servery umožňující webovou správu procesu

Otázka webových serverů co se týče problematiky správy procesů není tak důležitá, protože proces je spuštěn v kontaineru, který aplikační server nese. V podstatě každý jazyk, který implementuje rozhraní pro správu procesu a má svou webovou platformu, může být považován za vhodného kandidáta k vývoji rozhraní. V jazyku Java bude výsledný produkt po zkompilování nástrojem Maven balík *.war*, který je možno nasadit na každý aplikační server podporujici platformu J2EE.

Produkt	Vlastník	Java EE kompatibilita	Servlet specifikace	JSP specifikace
GlassFish	Oracle	5	2.5	2.1
JBoss	Red Hat	6	3.0	2.2
Jetty	Eclipse	částečná	3.0	2.1
WebLogic	Oracle	6	3.0	2.2
WebSphere	IBM	6	3.0	2.2
Tomcat	Apache	6	3.0	2.0

Tabulka 1: Java EE server specifikace

V tabulce aplikačních serverů 1 můžeme vidět, že všechny produkty jsou si do určité míry podobné a liší se především v licencích. Podpora Javy je na stejné úrovni, a tudíž

je pro naši aplikaci z hlediska nasazení výběr aplikačního serveru minoritní záležitostí. Vzhledem k faktu, že bude aplikace nositelem statusu otevřeného softwaru, rozhodl jsem se volit také aplikační server pod stejnou licencí, a to Apache Tomcat 5.6.

5 Technologie použité při vývoji a nasazení rozhraní

Dostáváme se do sekce blíže popisující technologie, ve kterých jsem se rozhodl rozhraní vyvíjet, nebo je použil k vývoji. Dále si přiblížíme i technologie, které byly použity pro nasazení softwaru. Každá sekce obsahuje stručný popis dané technologie a důvod pro zvolení k vytvoření aplikace. Obecně jsem volil pouze technologie, které jsou v dnešním světe IT považovány za jedny z nejlepších ve svém oboru.

5.1 Java

Technologie Java, viz [10] se skládá z programovacího jazyku Java a platformy Java. Programovací jazyk Java je vysokoúrovňový jazyk, který může být charakterizován následujícími výrazy:

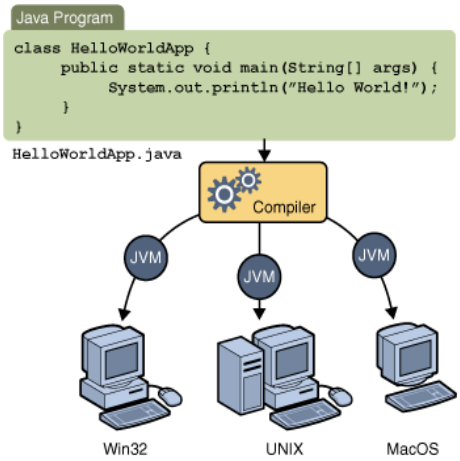
- Jednoduchý
- Nezávislý na architektuře
- Objektově orientovaný
- Přenosný
- Distribuovaný
- Vysoce výkonný
- Vícevláknový
- Robustní
- Dynamický
- Bezpečný

Každý z předcházejících výrazů je vysvětlen v The Java Language Environment, dokumentu, napsaném Jamesem Goslingem a Henrym McGiltnem.

V programovacím jazyce Java je všechen zdrojový kód nejdříve napsán v čistých textových souborech s příponou *.java*. Tyto zdrojové kódy jsou kompilovány do souborů s příponou *.class* javac kompilátorem. Soubor *.class* neobsahuje kód speciálně pro Váš procesor, místo toho obsahuje tzv. bytecode, což je kód určen pro Java Virtual Machine (Java VM). Nástroj java poté spustí Váš program jako instanci Java Virtual Machine. Protože je Java VM dostupný pro mnoho platforem *.class*, soubory mohou být spuštěny na různých platformách, jako je Microsoft Windows, SolarisTM Operating System (Solaris OS), Linux nebo Mac OS. Platforma Java se liší od ostatních platforem tím, že se jedná pouze o softwarovou vrstvu ležící nad hardwarově orientovanou platformou. Java platforma má dvě komponenty:

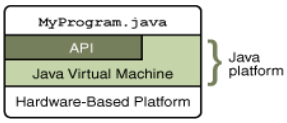
- Java Virtual Machine
- Java Application Programming Interface

Obrázek 5: Ukázka vývojového procesu v Javě



Obrázek 6: Beh java aplikace na spoustě různých platforem

Již jsme si přiblížili Java Virtual Machine: je to základ Java platformy, který je portován na mnoho operačních systémů. API je velká kolekce předem připravených částí kódů, které nabízí spoustu příjemných schopností. Je seskupena do knihoven příbuzných tříd a rozhraní. Tyto knihovny jsou známé jako balíčky (packages).



Obrázek 7: API a Java VM odstiňuje program od hardwaru

V zásadě je vysokoúrovňový programovací jazyk Java výkonnou platformou. Každá plná implementace Javy vám nabízí:

- **Vývojářské nástroje:** Vývojářské nástroje nabízí vše, co je potřeba pro kompilování, spouštění, monitorování, ladění a dokumentování aplikací. Pro Vás, jako nového vývojáře, budou základní nástroje, které budete používat *javac* kompilátor, *java* nástroj pro spouštění a dokumentační nástroj *javadoc*.

- **Application Programming Interface (API):**API poskytuje základní funkcionalitu programovacímu jazyku Java. Jedná se o velmi širokou kolekci tříd, určených pro okamžité použití ve Vašich aplikacích. Má veškeré potřebné komponenty od základních tříd přes podporu pro XML dokumentů, až po přístup k databázím, vzdálené volání metod a další. Jádro API je velmi rozsáhlé. Abyste se přesvědčili, odkazuji Vás na dokumentaci k Java SE Development Kit 7).

- **Distribuční (Deployment) technologie:** JDK software poskytuje standardní mechanismy jako je Java Web Start software a Java Plug-In software, pro distribuci Vašich aplikací k uživatelům.

- **Nástroje pro uživatelské rozhraní:** The Swing a Java 2D toolkits umožňují vytvářet lehce komplexní a sofistikovaná grafická uživatelská rozhraní (GUI).

- **Integrační knihovny:** Integrační knihovny, jako je např. Java IDL API, JDBC API, Java Naming and Directory Interface („J.N.D.I.“) API, Java RMI a Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology), jež umožňují přístup k databázím a volání vzdálených objektů.

Jazyk Java jsem si pro implementaci vybral hlavně z toho důvodu, že pracuji jako Software Developer zaměřený na Javu a Java based technologie, a jelikož Java patří v současnosti mezi nejpoužívanejší jazyky ve světě vývoje softwaru vůbec, je velice kompatibilní s různými druhy technologií a také precizně zdokumentována. Taktéž podpora této platformy a jazyka je vynikající.

5.2 Spring

Spring Framework, viz [11] je populární open-source aplikační rámec neboli framework (označován také jako kontejner) pro vývoj J2EE aplikací. Může být použit libovolnou Java aplikací a se stal populární v Java komunitě, hlavně jako alternativa k Enterprise Java Bean (EJB) nebo jako jeho nadstavba. Základním důvodem vzniku Spring Frameworku je usnadnění vývoje enterprise aplikací a to:

- Odstraněním těsných programových vazeb jednotlivých POJO objektů a vrstev, pomocí návrhového vzoru Inversion of Control.

- Možností volby implementace (EJB, POJO) business vrstvy pro aplikační architekturu, ne naopak (aby architektura předepisovala implementaci)

- Řešením různých aplikačních domén bez nutnosti použití EJB, například transakční zpracování, podpora pro remoting business vrstvy formou webových služeb či RMI.

- Podporou implementace komponent pro přístup k datům, ať už formou přímého JDBC či ORM (object-relation mapping) technologií a nástrojů jako je Hibernate, TopLink, iBatis nebo JDO.

- Odstraněním závislosti na roztroušených konfiguracích a dohledávání jejich významu.
- Abstrakcí vedoucí ke zjednodušenému používání dalších části J2EE, jako například JMS, JMX, JavaMail, JDBC, JCA nebo JNDI.
- Usnadněním používání a psaní unit testů.
- Správou a konfiguračním managementem business komponent.

Spring Framework se skládá z částí organizovaných do přibližně dvaceti modulů. Moduly jsou rozděleny do skupin:

- Core Container - skládá se z modulů Core, Beans, Context a Expression Language.
  - Moduly **Beans** a **Core** poskytují základní části frameworku, včetně IoC a Dependency Injection. BeanFactory je sofistikované provedení návrhového vzoru factory (továrna).
  - Modul **Context** staví na pevném základu poskytnutém moduly Core a Beans. Je to prostředek pro přístup k objektům způsobem, který je podobný rozhraní JNDI. Modul Kontext dědí vlastnosti z modulu Beans a přidává podporu pro internacionalizaci a Java EE funkce jako jsou EJB a JMX. Ústředním bodem modulu Context je rozhraní ApplicationContext.
  - Modul **Expression** je rozšíření jednotného jazyka pro vytváření výrazů (unified expression langure), jak je uvedeno ve specifikaci JSP 2.1.
- Data Access/Integration - skládá se z JDBC, ORM, OXM, JMS a modulu Transactions.
  - Modul **JDBC** poskytuje abstraktní JDBC vrstvu, která odstraňuje potřebu dělat nudné JDBC kódování a zachytávání chybových kódů specifických pro danou databázi.
  - Modul **ORM** poskytuje integrační vrstvy pro populární API objektově relačního mapování, včetně SPS, JDO, Hibernate a iBatis. Pomocí ORM balíčku můžete využít všech uvedených frameworků pro O/R mapování v kombinaci se všemi ostatními funkcemi, které SpringFramework nabízí, jako je například jednoduchý nástroj na řízení transakcí.
  - Modul **OXM** poskytuje abstraktní vrstvu, která podporuje implementaci Object/XML mapování pro JAXB, Castor, XMLBeans, JiBX a Xstream.
  - Modul **JMS** (Java Messaging Service) obsahuje funkce pro tvorbu a příjem zpráv.
  - Modul **Transactions** podporuje programové a deklarativní řízení transakcí pro třídy implementující speciální rozhraní a pro všechny Vaše POJO (Plain Old Java objekty).

- Web - skládá se z modulů Web, Web-Servlet, Web-Struts, a Web-portlet.
  - Modul **Web** ve Spring Frameworku poskytuje základní, webově orientované funkce, jako je funkce multipart file-upload.
  - Modul **Web-Servlet** obsahuje Springovou implementaci model-view-controller (MVC) pro webové aplikace.
  - Modul **Web-Struts** obsahuje podpůrné třídy pro integraci Struts do webových aplikací v rámci Springu.
  - Modul **Web-Portlet** umožňuje implementaci MVC, které mají být použity v prostředí portletu a zrcadlí funkce Web-Servlet modulu.
- AOP (Aspect Oriented Programming) a Instrumentation - AOP je modul implementující podporu pro aspektově orientované programování. Umožňuje separování části kódu prolínající se celou aplikací (autorizace, logování, transakce) do takzvaných aspektů a jejich následnou aplikaci na jakýkoli POJO objekt. Využití AOP modulu se prolíná celým frameworkem a jedná se o jednu z nejsilnějších vlastností Springu.
- Test - podporuje testování komponent Springu pomocí JUnit nebo TestNG.

Spring framework jsem si pro vývoj vybral především z těchto důvodů:

- Spring MVC Modul - pokrývá veškeré aspekty vzoru Model-View-Controller používaného pro vývoj webových aplikací. Spolu s anotacemi tvoří velice silný nástroj pro vývoj webových komponent.
- Spring Security Modul - modul, na kterém jsem stavěl autentizaci uživatelů. Výhodou je přímočará konfigurace a možnost přistupovat k uživatelské relaci velice triviálním způsobem, bez další implementace. Také již naimplementovaná servisní část zajisté uspokojí uživatele.
- Správa a konfigurační management - aplikace obsahuje konfigurační soubor pro databáze, multithreading, a mnoho dalších aspektů. Uživateli neznalému frameworku a javy dává možnost lehce nakonfigurovat tyto hodnoty bez jakéhokoli zásahu do kódu nebo vnitřní aplikační struktury.
- Snadná konfigurace projektové struktury
- Snadná konfigurace lokalizace
- Web Modul a jeho funkce multipart file-upload používaná k uploadovaní souborů na server

5.3 JPA

Java Persistence API (JPA), viz [12] je framework programovacího jazyka Java, který umožňuje objektově relační mapování (ORM). To usnadňuje práci s ukládáním objektů do databáze a naopak. Je určen jak pro Java SE, tak pro Java EE. Entita je objekt, který reprezentuje data v databázi. Typicky entitní třída reprezentuje tabulku v relační databázi a každá instance této třídy pak koresponduje k jedné řádce tabulky.

Aby mohly být entity persistovány, musí mít entitní třída následující vlastnosti:

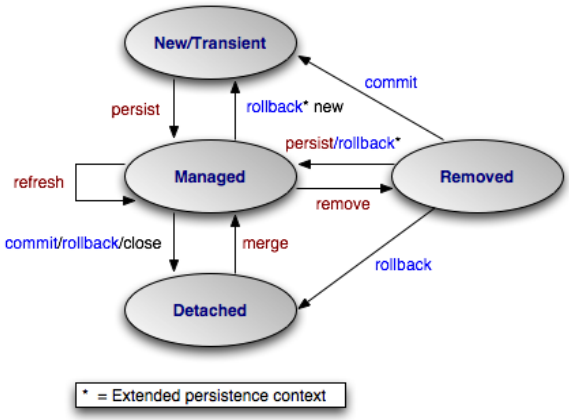
- Musí být anotována anotací *@Entity*
- Musí mít public nebo protected konstruktor bez parametrů. Může mít ale i další konstruktory
- Nesmí být deklarována jako final. To platí i pro její metody
- Pokud session beana bude pracovat s instancemi této třídy a bude typu Remote, pak tato entitní třída musí implementovat interface Serializable
- Může dědit z entitní i ne-entitní třídy
- Její atributy musí být deklarovány jako private, protected nebo package-private a lze k nim přistupovat pouze přes metody (gettery a settery)

Aby mohla být entita uložena do databáze, musí být atributy pouze následujících typů:

- primitivní typy
- java.lang.String
- Wrapper třídy primitivních typů
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.TimeStamp
- byte[]
- Byte[]

- char[]
- Character[]
- Enumerační typy
- Jiné entity a/nebo kolekce entit
- Třídy s anotací Embeddable

Každá entita má svůj určitý stav, ve kterém se nachází. Ten je rozhodnut instancí třídy EntityManger. Entita se pohybuje mezi jednotlivými stavy vždy po provedení určité akce EntityMangerem. Startovním stavem entity je vždy New/Transient, žádný stav není konečný.



Obrázek 8: Stavy entity a hrany mezi nimi [13]

Existují 4 typy multiplicit v entitních třídách:

- **One-to-one:** instance má referenci na jednu entitu jiné třídy.
- **One-to-many:** instance má reference na množinu instancí jiné třídy.
- **Many-to-one:** více instancí má referenci na jednu instanci jiné třídy.
- **Many-to-many:** více instancí má reference na instance jiné třídy.

Entity jsou spravovány objektem třídy EntityManager. Chceme-li ho získat v SessionBeaně, musíme deklarovat atribut typu EntityManager a anotovat ho pomocí Persistence Context. Další možností je definovat například Hibernate Transaction Managera ve Springu a dát mu referenci na sessionFactory definovanou také ve springu. Transakci pak můžeme spravovat velice jednoduše pomocí anotace *@Transactional* přímo v servisní třídě.

Také upřednostnění ORM frameworku, konkrétně JPA má v naší implementaci důvod. Především lehká a přehledná práce s objekty, které zastupují jak uživatele, tak scénaře, testy, položky testů a také vytvoření DB tabulek dle schématu nadefinovaného v daných doménových třídách, díky kterým je správce oprošten o nutnost vytvářet tyto struktury sám. U Springu i u JPA používám anotace místo XML konfigurace pro lepší orientaci a správě kódu.

5.4 Maven

Maven je nástroj pro správu, řízení a automatizaci buildů aplikací. Je to velice mocný nástroj. Z našeho pohledu jsou jeho hlavní výhody tyto:

- jeho formát projektu je akceptován všemi vývojovými prostředími (IntelliJ IDEA, Eclipse, NetBeans)
- poskytuje repository obsahující víceméně všechny javové knihovny ve všech verzích, mnoho z nich včetně zdrojových kódů a javadocu
- v základním nastavení nás nutí dodržovat osvědčené konvence a postupy

Výhoda univerzálního formátu projektu je zřejmá. Můžeme používat libovolné vývojové prostředí nebo například editor vi a příkazový řádek. Formát projektu je stále totožný. Repository nás zbavuje nutnosti starat se o knihovny a jejich požadavky, protože v repository má každá knihovna uvedeny i další knihovny, na kterých je závislá. Maven má přednastavené postupy, které se v praxi osvědčily. Odděluje kupříkladu zdrojové kódy tříd (ty jsou v adresáři *src/main/java*) od zdrojových kódů unit testů (jsou v adresáři *src/test/java*), i ostatní soubory typu obrázky má odděleně, konfigurační soubory, properties a další (ty jsou v adresáři *src/main/resources*). Obvyklé operace jsou přednastaveny, takže např. příkazem `mvn package` dáme povel k zabalení výsledků projektu, což zahrnuje kompilaci, spuštění unit testů a zabalení do souboru JAR nebo WAR [14].

Každý projekt má svůj konfigurační soubor *pom.xml* 21 (POM jako Project Object Model). Tento soubor obsahuje strukturu projektu v XML formátu a definuje jednotlivé části a závislosti projektu na externích knihovnách. Závislost lze nastavit pomocí tagu dependency, čímž nás oprosťuje od nutnosti stahovat všechny balíky a importovat je jako knihovny do struktury projektu manuálně. Obsahuje také řadu užitečných pluginů, které nám umožňují například generovaní tříd pro práci s webservisy na základě WSDL souboru nebo generování java souboru z XSD tzv. JAX-B.

Maven je na vývoj použit z prostého důvodu. Lepší buildovací nástroj nejen na java aplikace v současné době najdeme jen stěží. Dále je nedílnou součastí všech velkých projektů různých korporací.

5.5 Oracle XE

Oracle Express Edition (XE) je databáze založená na jádru Oracle Database 11. S Oracle Database XE. Můžete vyvíjet a nasazovat aplikace, infrastruktury, poté upgradovat bez ná-

kladného a složitého "stěhování", v případě potřeby. Důležitou vlastností Express Edition je to, že sdílí s ostatními edicemi stejné jádro databázového serveru, včetně mechanismů řízení současného přístupu, které umožňují efektivně zvládat vysoké počty transakcí s minimem vzájemného blokování. Základem je zamykání záznamů na úrovni řádků a tzv. Multi-Version Read Consistency, mechanismus, který zajišťuje, že nedochází k vzájemnému blokování zápisových a čtecích operací.

Limity a omezení:

- Limit objemu uživatelských dat: 11 GB
- Velikost použité operační paměti: 1GB
- Pouze Linux a Windows
- Verze x86(x32) má pouze APEX verze 2.1
- Verze x64 má pouze APEX verze 4.0
- Nefunguje HTTPS (místo Apache serveru je totiž použit Oracle Net Listener)
- Může být instalován na více CPU serverů, avšak může být spuštěn pouze na jeden procesor v každém serveru
- Pouze jedna instance na fyzickém PC
- Není omezen počet uživatelů, kteří jsou připojeni najednou

Oracle XE jsem si vybral z důvodu velice snadné instalace, především oproti tradiční plnohodnotné Oracle databázi, a taktéž vhledem k dostupnosti produktu. Samozřejmě Oracle XE je také postačující kandidát z databázového pohledu naší aplikace z důvodu minimální zátěže databáze, a to i při vysokém počtu současně probíhajících testů.

5.6 Apache Tomcat

Apache Tomcat je nejpoužívanější open source web server a servlet kontejner, který je vytvářen nadací Apache Software Foundation. Tomcat implementuje dvě Java EE specifikace:

- Servlety
- JavaServer Pages (JSP)

a poskytuje Java HTTP web server prostředí ke spuštení kódu. Servlet container implementující výše zmíněné specifikace se nazýva Catalina. Další modul Coyote slouží jako HTTP Connector podporující HTTP 1.1 protokol pro aplikační kontejner nebo web server. V podstatě nám poslouchá na TCP portu a přeposílá requesty na jádro tomcatu a poté je zpracované posílá zpět klientovi. Další komponenta je Jasper, který slouží jako parser JSP filů, které kompiluje do Java kódu jako servlety. Tomcat má mnoho dalších

Java EE specifikace	Tomcat verze	vyžaduje verzi Java SE
Java EE 6	7.0.x	Java 6
Java EE 5	6.0.x	Java 5
J2EE 1.4	5.5.x	J2SE 1.4

Tabulka 2: Java EE specifikace pro Tomcat

komponent, jako kupříkladu Cluster, zabezpečující loadbalancing. Každá verze Apache Tomcat implementuje jinou verzi Java EE specifikace.

Tomcat jsem si vybral jako web server ze 4 důvodů:

- jednoduchost
- transparentnost
- menší náročnost než konkurenční servery
- rychlejší vývoj

5.7 Netbeans

Netbeans je IDE vyvíjeno pod hlavičkou Sun Microsystems a v současnosti pod správou Oracle. Je kompletně napsáno v Javě a také je primárně určeno pro vývoj v jazyce Java, ale podporuje i další programovací jazyky jako je C++, PHP, Ruby. Netbeans je také multiplatformní jak na systémové úrovni, tak i mezi hlavními Java platformami(J2SE, J2ME, J2EE).

Po dlouholeté zkušenosti s různými IDE dávám přednost Netbeansu při práci s javou, protože je lépe přizpůsoben pro mavenizované projekty a celková orientace ve struktuře je velice user-friendly.

6 Vývoj rozhraní

Vývoj rozhraní probíhal v iteratívním modelu. Po specifikaci požadavků diplomové práce jsem tudíž začal s analýzou a návrhem následovanou implementací a nasazením na aplikační server. V následujících kapitolách nahlédneme na jednotlivé procesy vývoje aplikace z hlediska programátora a uvidíme jejich nejdůležitější části také v podobě různých zdrojových kódu, use-case diagramů a databázových schémat.

6.1 Požadavky

6.1.1 Funkční požadavky

1. **Autentizace uživatele**
   Vzhledem k možnosti DoS útoku prostřednictvím tohoto nástroje by mělo být umožněno, aby správce zajistil přístup pouze autentizovaným uživatelům. Autentizaci lze řešit přes Apache, další možností je použít implementaci přímo na bázi aplikační.

2. **Volba režimu**
   V zásadě je třeba oddělit režim spouštění testů a prohlížení výsledků. Rozhraní by proto mělo nabízet rozcestník do těchto dvou vzájemně nezávislých větví.

3. **Volba testu**
   Je potřeba implementovat rozhraní pro 2 testy - registrace a volání. Uživateli by měla být nabídnuta možnost volby, ať už prostřednictvím některé grafiky nebo prostřednictvím dynamického formuláře. Po volbě testu se uživateli objeví formulář pro zadání uživatelských parametrů:

   - v případě hovorů dva totožné formuláře pro UAC a UAS, pro registrace stačí jen jeden
   - zdrojové SIP sockety pro UAC(generátor) a UAS (posluchač) testu (tj. umožnit více provozů na jedné IP adrese z různých portů a zároveň umožnit i provozy z různých IP)
   - cílová IP adresa/socket SIP serveru
   - délka hovoru
   - maximální počet souběžných session
   - možnost výběrů scénáře (Uac nebo Uas)
   - možnost výběrů csv souborů s daty obsahující detailní informace o hovoru

4. **Výpočet parametrů pro jednotlivé procesy**

   - počet procesů UAC a UAS ze zadaného počtu kombinací SIP socketů
   - počet hovorů generovaných jedním procesem UAC
   - maximální počet hovorů pro jeden proces

- maximální počet souběžných hovorů pro jeden proces
- rozpočítat jednotlivé procesy mezi procesorová jádra

5. **Spuštění a dohled nad testem**
   Uživateli by mělo být umožněno vidět aktuální průběh testu, tj. buďto z stdout procesů sipp (velmi obtížné), nebo z generovaných csv souborů updatovat jednak databázi a jednak data v grafickém rozhraní. O ukončení testu je třeba uživatele informovat a také se pojistit proti zatuhnutí aplikace sipp. Pokud se po nějaké době sama korektně neukončí, je třeba ji killnout.

6. **Vložení dat do databáze**
   V průběhu testu nebo po něm, je třeba vložit do databáze parametry testu, testové výsledky z csv souborů a chyby z vygenerovaných textových souborů tak, aby uživatel mohl procházet jednotlivé testy a jejich výsledky zpětně. V případě smazání jednoho záznamu, např. záznamu o chybách, by měl být smazán celý test. Popřípadě umožnit mazání jen prostřednictvím některé hlavní tabulky. Výsledky by měly jít zobrazit i graficky.

6.1.2 Nefunkční požadavky

- intuitivní spouštění testových procesů
- využití databáze Oracle
- spolehlivost
- umístění na serveru VŠB-TUO
- zajištění ochrany osobních dat
- optimalizace pro prohlížeče IE, FF, Chrome

6.2 Analýza a Návrh

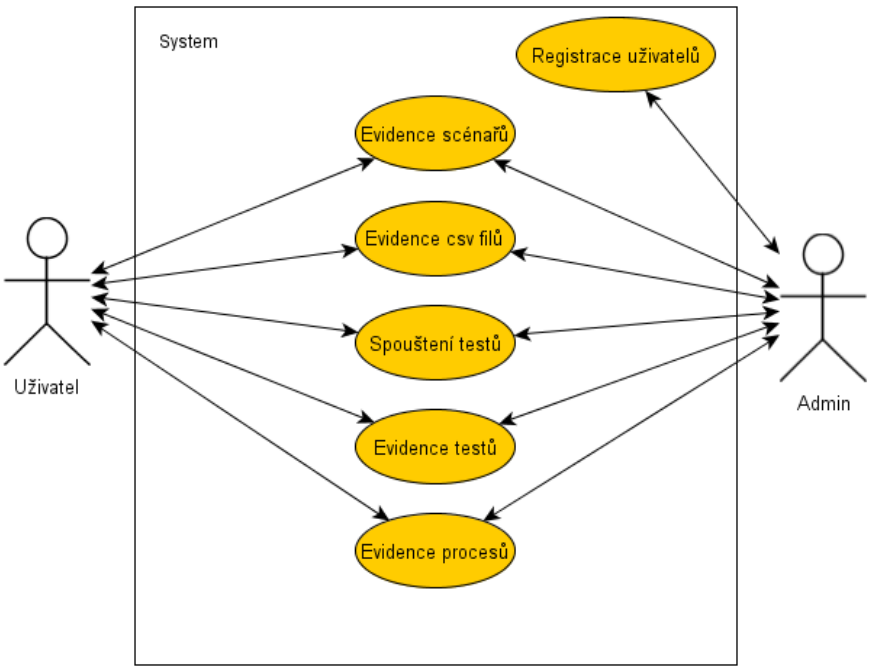
Tato sekce obsahuje komplexní architekturu vytvářeného systému a poskytuje přehled jak prezenční, databázové, tak i aplikační vrstvy.

6.2.1 Use-Case pohled (případy užití)

1. **Hlavní diagram užití**
   Tento diagram 9 popisuje základní případy užití v naší budoucí aplikaci.

   Aktéři:

   - *Admin* – provádí správu celého systému, vytváří nové uživatele, provádí údržbu systému



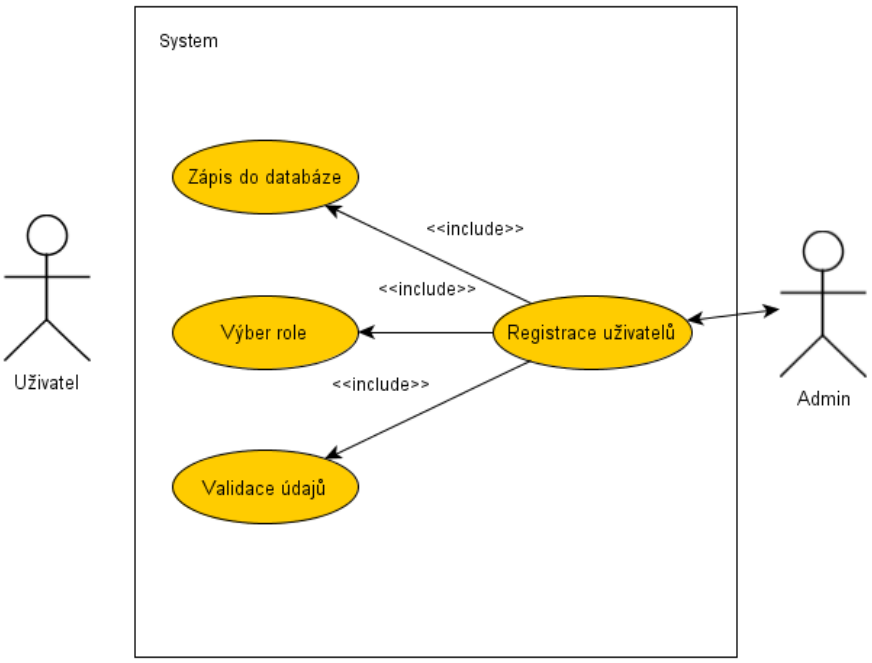
Obrázek 9: Základní případy užití v systému

- *Uživatel* - bežný účastník, který může spouštět a spravovat testy, vytvářet si své portfolio scénařů a csv souborů

Aktivity:

- *Registrace uživatelů* – zde se registrují jednotliví účastníci systému údržbu systému
- *Evidence testů* - aktivita sloužící k evidenci UAC nebo UAS testů, jejich vytváření, přehledu a mazání.
- *Evidence scénařů* - aktivita sloužící k evidenci UAC nebo UAS scénařů, jejich vytváření, přehledu a mazání.
- *Evidence csv filů* - aktivita sloužící k evidenci externích csv souborů obsahujících detailní data o hovoru, jejich vytváření, přehledu a mazání.
- *Spouštení testů* - aktivita sloužící ke spouštení UAC nebo UAS testů a vytvoření procesních statistik v databázi.
- *Evidence procesů* - aktivita sloužící ke správě testových procesů

2. **Registrace účastníků**
Registrace 10 popisuje proces registrace jednotlivých účastníků. Registraci provádí pouze osoba v roli admina, a to z důvodu, aby aplikace nemohla být zneužita jako nástroj pro uskutečnění Dos útoku.



Obrázek 10: Registrace uživatelů

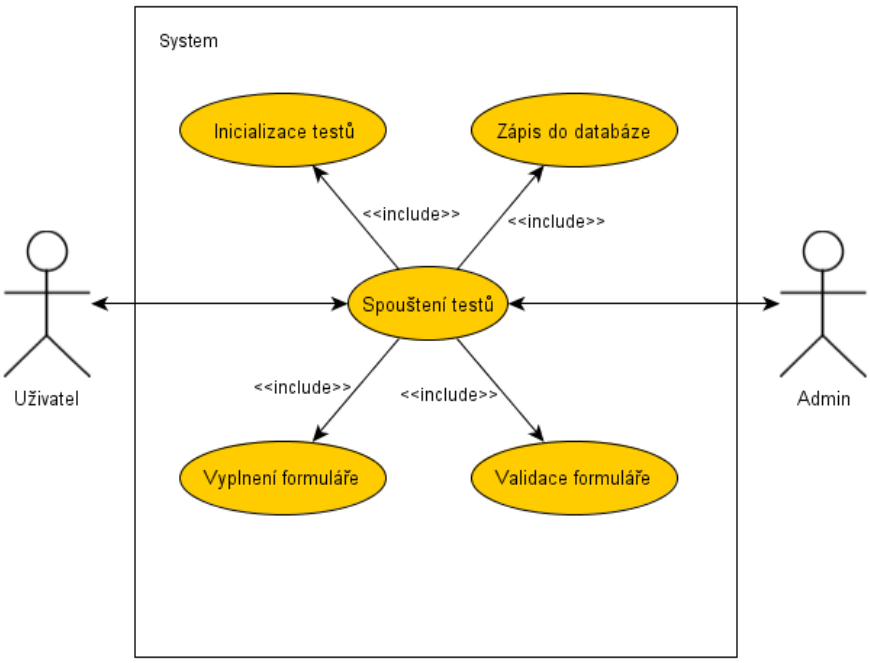
Aktivity:

- *Registrace uživatelů* – tato aktivita se skládá z těchto dílčích částí
  - *Zápis do databáze* - vytváří v databázi záznam s daty o registrovaném uživateli
  - *Výber role* - při registraci osoba z admin právy nastavuje příslušnou roli pro uživatele
  - *Validace údajú* - před potvrzením registrace jsou údaje zvalidovány

3. **Spuštení testů**
   Spouštení testů 11 slouží k zadání testových parametrů a k následnému spuštení ať už jednosměrného či obousměrného testu.

   Aktivity:

   - *Spouštení testu* – se skládá z těchto částí
     - *Vyplnení formuláře* - slouží k vyplnění formuláře s hodnotami testu, pro které chceme příslušný test spustit
     - *Validace formuláře* - než odešleme test na inicializační modul, je zapotřebí ověřit, zda-li jsou poskytnutá data validní
     - *Inicializace testů* - z dat zadaných do formuláře vytvoří inicializační modul testy a poté je spustí



Obrázek 11: Spouštení testu

– *Zápis do databáze* - informace o testovém procesu jsou uloženy do databáze

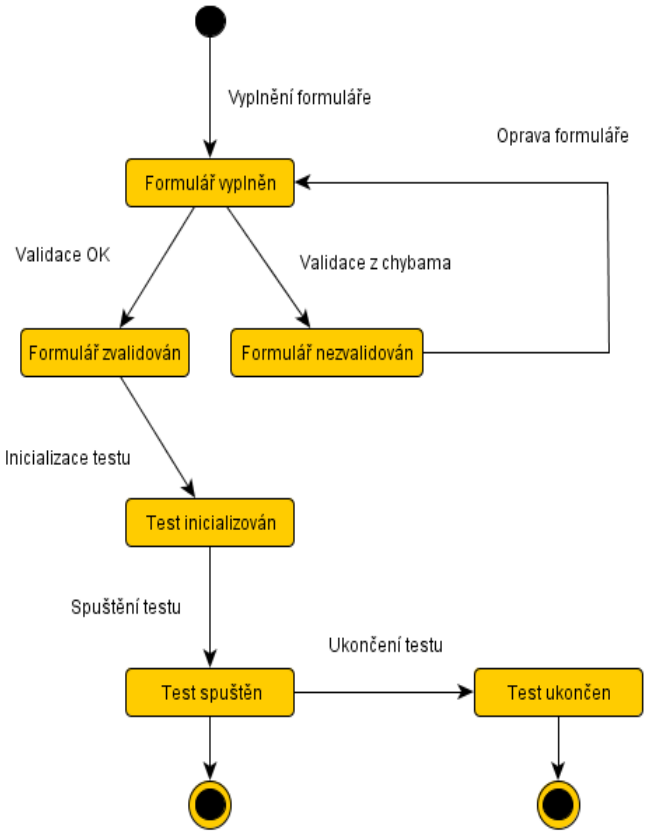
Další use-case diagramy, které jsou svou funkcionalitou téměř totožné:

- *Evidence testů*
- *Evidence scénářů*
- *Evidence csv filů*
- *Evidence procesů*

Každá položka se skladá z vytváření, přehledu a mazání daných entit.

6.2.2 Stavový diagram

Stavový diagram 12 popisuje průběh testu od vyplnění formuláře po následnou validaci a spuštění. Pokud formulář obsahuje chyby, uživatel je vyzván k opravě a revalidaci před spuštěním. Po úspešné validaci probíhá inicializace testů a to tak, že je vytvořený multivláknový příkaz na spuštění dílčích testů v závislosti od počtu jader procesoru stroje, na kterém je aplikace provozována. Následuje spuštení testů s možností testy dokončit, pokud nechceme vyčkávat na řádné ukončení procesů.



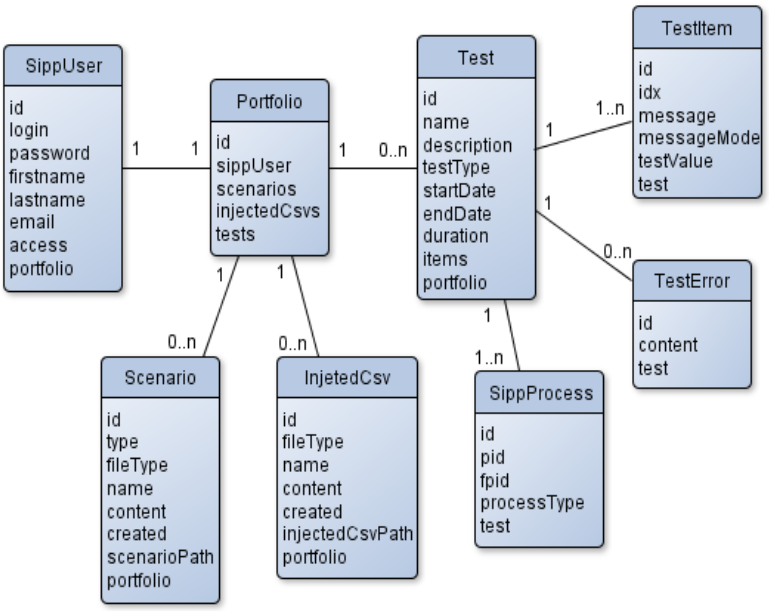
Obrázek 12: Stavový diagram popisující spuštění testu

6.2.3 Databázové schéma

Databáze obsahuje celkem 8 tabulek 13:

- *SippUser* - obsahuje informace o uživatelích systému
- *Portfolio* - každý uživatel má své portfólio, ke kterému se vážou testy, seznam scénářů, externích csv souborů a další, a to z toho důvodu, aby bylo možné snadno naprogramovat *actAs* funkcionalitu do systému i v budoucnu bez velkých zásahů do databázového schématu.
- *Test* - obsahuje záznamy o jednotlivých testech jako je typ testu, čas provádení testu
- *SippProcess* - obsahuje informace o systémových procesech k danému testu
- *Scenario* - ukládá a spravuje scénáře nahrané uživatelem do systému
- *InjectedCsv* - ukladá a spravuje připojené csv soubory nahrané do systému

- *TestItem* - obsahuje seznam testových položek/výsledků k danému testu
- *TestError* - obsahuje seznam problémů, které se vyskytly v době vykonávání testů



Obrázek 13: Databázová schéma aplikace

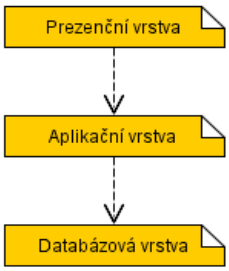
6.2.4 Logický náhled

Na následujícím obrázku 14 můžeme vidět logický pohled na aplikaci, skládající se z následujících vrstev:

- *Prezenční vrstva* – zobrazuje a formátuje uživateli informace – implementace v JSP, jQuery, HTML a CSS
- *Aplikační vrstva* - provádí logiku – implementace v Javě za použití frameworku Spring
- *Databázová vrstva* - manipulace a uchování dat - implementace v JPA a Hibernate, použitá databáze Oracle XE

6.2.5 Procesní náhled

Základní procesy, které budou běžet v naší aplikaci, jsou založeny především na práci se scénaři, csv soubory a testy. Pro všechny uvedené procesy, aplikace zabezpečuje operace vytvoření, smazání a samozřejmě i zobrazení. U testů také sadu operací na ovládání

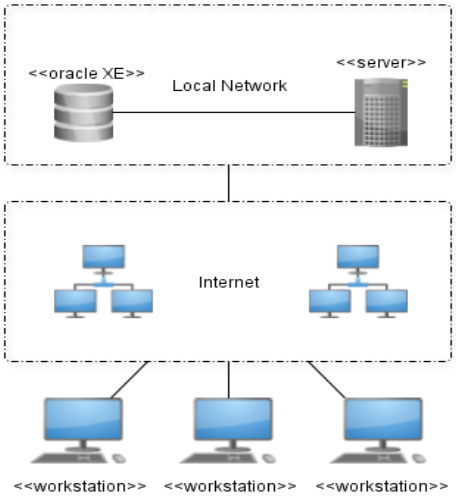


Obrázek 14: Logický pohled

testových procesů, které jsou z hlediska výkonu nejnáročnějším článkem aplikace. Aby uživateli byly umožněno jednotlivé procesy spouštět, musí prvně úspešně projít autentizací systému, která je postavena na modulu *Spring Security*.

6.2.6 Náhled rozmístění

Aplikace běží na lokální síti společně s databází, přičemž doména, na které je nasazená, má veřejnou Ip adresu čili je volně dostupná z internetu. Uživatel má tedy možnost připojit se a začít testovat odkudkoliv na světě, poté co je uživateli vytvořen účet k aplikaci. Jako aplikační server byl vybrán Apache Tomcat a databázový server Oracle XE, jak už jsem blíže specifikoval v sekci technologie.



Obrázek 15: Náhled rozmístění

6.2.7 Výkon a kvalita

Výkon aplikace závisí zejména na počtu paralelně spuštených testů a serveru, na kterém aplikace poběží. Všeobecně bude platit, čím větší počet jader procesoru, tím vyššího výkonu při testování sip infrastruktury sítě dosáhneme, jelikož databázové operace aplikace zabírají minimum vypočetního výkonu. Aplikace je navržena tak, aby byla podporována standartními prohlížeči a to:

- Mozilla Firefox ve verzi 18 a výše
- Google Chrome ve verzi 18 a výše
- Internet Explorer ve verzi 8 a výše

6.3 Implementace

Jelikož se jedná o webové rozhraní, byl pro implementaci rozhraní použit modul Spring MVC, který představuje softwarovou architekturu, jež rozděluje datový model aplikace řídící logiku a uživatelské rozhraní aplikace do tří nezávislých komponent tak, aby modifikace některé části měla minimální vliv na ostatní. Komponenty MVC architektury:

- **Model (Model)** Doménová reprezentace dat v aplikaci a business logika, která s daty pracuje
- **View (Pohled)** Převádí data z doménových objektů do podoby vhodné k prezentaci
- **Controller (Řadič)** Reaguje na události (typicky od uživatele) a na jejich základě zajišťuje změny v modelu, ale i v pohledu.

Spring nám nabízí konfiguraci přes xml nebo anotace. K vytvoření aplikace jsem volil anotace, jelikož má programátor možnost konfigurovat aplikaci přímo ve zdrojovém kódu a je oprošten od udržování separátních xml souborů. Minimální konfigurace v xml pro hibernate, spring-security, jsp-resolver, lokalizaci, resource-mapping byla nezbytná. Pro lepší přehled jsem konfiguroval v xml taktéž objekty, které používají strategy pattern jako argument konstruktoru. Poslední konfigurace v xml je věnována tzv. dekorátorům, kteří se starají o vzhled sránky a zabraňují opakování zdrojového kódu z hlediska pohledů.

6.3.1 Model (Model)

Model sekce se skladá z dvou podsekcí, které si zde blíže rozebereme:

- **Domain classes (Doménové třídy)** - zastupují všechny entity, které vystupují v aplikaci a starají se za pomoci ORM technologií (Hibernate + JPA), aby data v aplikaci byly perzistentní a namapovány na příslušnou databázovou tabulku v dané databázové struktuře.

- **Data access object (DAO třídy)** - starají se o databazové operace z hlediska doménových tříd
- **Service classes (Servisní třídy)** - zabezpečují business logiku, která pracuje s doménovými třídami.

Doménové třídy

- *SippUser 4* - třída SippUser je určena k uchování dat o uživateli systému. Je součástí třídy *SippUserDetails*, která slouží na uchování uživatelské relace.

```
@Entity
@Table(name = "SIPP_USER")
public class SippUser implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "USER_ID")
    @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;
    @Column(name = "LOGIN")
    private String login;
    @Column(name = "PASSWORD")
    private String password;
    @Column(name = "NAME")
    private String firstname;
    @Column(name = "SURNAME")
    private String lastname;
    @Column(name = "EMAIL")
    private String email;
    @Column(name = "SECURITY_FLAG")
    private boolean access;
    @OneToOne(mappedBy = "sipuser", cascade = CascadeType.REMOVE)
    private Portfolio portfolio;
}
```

Výpis 4: SippUser.java bez gettrů a settrů

- *Portfolio* - třída Portfolio funguje jako entita sjednocující dílčí prvky aplikace do jednoho profilu a navazuje na konkrétního uživatele. Tato vazba má do budoucna zajistit vytváření nových funkcí pro uživatele, jako je například dříve zmíněný *actAs*
- *Scenario* - třída Scenario reprezentuje scénaře ať už obousměrných nebo jednosměrných testů. Daný xml soubor, který scénář reprezentuje, je také uchován na severu v profilu uživatele.
- *InjectedCsv* - třída InjectedCsv reprezentuje na rozdíl od třídy Scenario csv soubory s detailními daty o volaném účastníkovi.
- *Test 5* - třída Test uchovává informace o průběhu testu.

```
@Entity
@Proxy(lazy=false)
@Table(name = "SIPP_TEST")
public class Test {

    @Id
    @Column(name = "TEST_ID")
    @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;
    @Column(name = "NAME")
    private String  name;
    @Lob
    @Column(name = "DESCRIPTION")
    private String  description;
    @Column(name = "IS_ACTIVE")
    private Boolean active;
    @Column(name = "IS_FINISHED")
    private Boolean finished;
    @Enumerated(EnumType.STRING)
    @Column(name = "TEST_TYPE")
    private TType testType;
    @Column(name = "START_DATE")
    @Temporal(javax.persistence.TemporalType.TIMESTAMP)
    private Date startDate;
    @Column(name = "END_DATE")
    @Temporal(javax.persistence.TemporalType.TIMESTAMP)
    private Date endDate;
    @OneToMany(mappedBy = "test", cascade = CascadeType.REMOVE, fetch = FetchType
        .EAGER)
    private Set<TestItem> items;
    @OneToMany(mappedBy = "test", cascade = CascadeType.REMOVE, fetch = FetchType
        .EAGER)
    private List<SippProcess> processes;
    @ManyToOne
    @JoinColumn(name = "PORTFOLIO_ID")
    private Portfolio   portfolio ;
}
```

Výpis 5: Test.java bez gettrů a settrů

- *TestItem 6* - třída TestItem je nositel dílčích informací o testu a obsahuje statistiky zpráv

```
@Entity
@Table(name = "SIPP_TEST_ITEM")
public class TestItem {

    @Id
    @Column(name = "TEST_ITEM_ID")
    @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;
    @Column(name = "IDX")
    private String  idx;
```

```
    @Column(name = "MESSAGE")
    private String message;
    @Column(name = "MESSAGE_MODE")
    private String messageMode;
    @Column(name = "TEST_VALUE")
    private int testValue;
    @ManyToOne
    @JoinColumn(name="TEST_ID")
    private Test test;
}
```

Výpis 6: TestItem.java bez gettrů a settrů

- *SippProcess* - informací o systémových procesech k danému testu
- *SippUserDetails 7* - nositel informací z aktuální uživatelské relace

```
public class SippUserDetails extends User {

    private SippUser sippUser;

    public SippUserDetails(String username, String password, boolean enabled, boolean
        accountNonExpired, boolean credentialsNonExpired, boolean accountNonLocked,
        Collection<? extends GrantedAuthority> authorities, SippUser sippUser) {
        super(username, password, enabled, accountNonExpired, credentialsNonExpired,
            accountNonLocked, authorities);
        this.sippUser = sippUser;
    }

    public SippUser getSippUser() {
        return sippUser;
    }
}
```

Výpis 7: SippUserDetails.java

Jak můžeme vidět, všechny doménové třídy s ORM jsou anotovány jako *@Entity* a pomocí anotace *@Table* jsou zmapovány na příslušnou tabulku v databázi. Jednotlivé atributy doménové třídy představují sloupce v dané tabulce s příslušnými parametry, které reprezentují anotace(*@Column*, *@Id*, *@OneToMany*, ...).

DAO

- *UserDAO*
- *PortfolioDAO*
- *ScenarioDAO 8*

```
@Repository
public class ScenarioDAO {
```

```
    @Autowired
    private SessionFactory sessionFactory;

    public void createScenario(Scenario scenario) {
        sessionFactory.getCurrentSession().save(scenario);
    }

    public List<Scenario> listScenario(Portfolio portfolio ) {
        Query query = sessionFactory.getCurrentSession().createQuery("from Scenario u
            where u.portfolio=:portfolio");
        query.setParameter("portfolio", portfolio );
        return query.list();
    }

    public void deleteScenario(Scenario scenario) {
        sessionFactory.getCurrentSession().delete(scenario);
    }

    public Scenario findScenarioById(Long id) {
        return (Scenario) sessionFactory.getCurrentSession().load(Scenario.class, id);
    }
}
```

Výpis 8: ScenarioDAO.java

- *InjectedCsvDAO*
- *TestDAO*
- *SippProcessDAO*

Povšimněme si, že DAO třídy jsou anotovány jako *@Repository* a pomocí anotace *@Autowired* si dokáží z xml konfigurace přivést potřebnou *sessionFactory 9*, aby mohli pracovat s databází.

```
<!-- Hibernate session factory conf -->
 <bean id="sessionFactory" class="org.springframework.orm.hibernate3.
      LocalSessionFactoryBean">
     <property name="dataSource" ref="dataSource" />
     <property name="configLocation" value="classpath:hibernate.cfg.xml" />
     <property name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" />
     <property name="hibernateProperties">
         <props>
             <prop key="hibernate.dialect">${jdbc.dialect}</prop>
             <prop key="hibernate.show_sql">true</prop>
             <prop key="hibernate.hbm2ddl.auto">update</prop>
         </props>
     </property>
 </bean>
```

Výpis 9: spring-servlet.xml Hibernate

Servisní třídy

- *UserService*

```
public interface UserService {

    public void createUser(SippUser user);
    public Paginator listUser(int pageNumber);
    public List<SippUser> listUser();
    public void deleteUser(SippUser user);
    public SippUser findUserById(Long id);
    public SippUser findUserByLogin(String login);
    public void createUserFolder(String login);
    public void deleteUserFolder(String login);
}
```

Výpis 10: UserService.java

- *PortfolioService*
- *ScenarioService*

```
public interface ScenarioService {

    public void uploadScenario(MultipartFile file, String type);
    public Paginator listScenario(int pageNumber);
    public List<Scenario> listScenario();
    public Scenario getScenario(Long id);
    public void deleteScenario(Long id);
    public void createScenarioFolder();
}
```

Výpis 11: ScenarioService.java

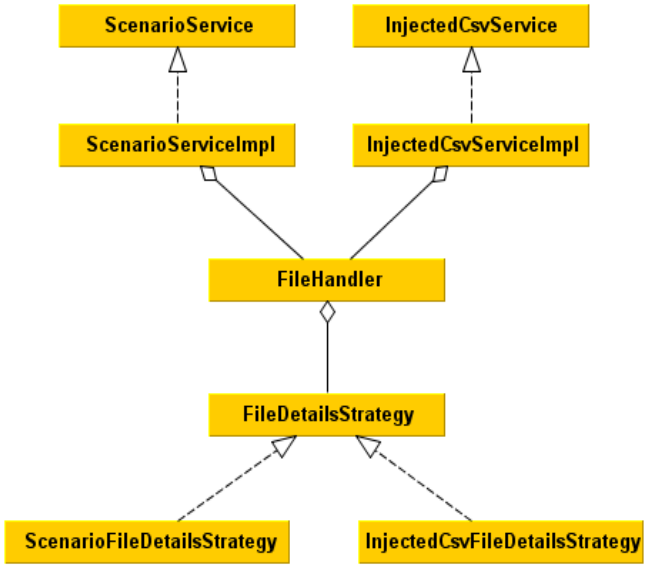
- *InjectedCsvService*
- *TestService*

```
public interface TestService {

    public void updateTest(Test test, List<Entry<String>> testData);
    public Test createTest(TType type);
    public void updateTests(List<Test> tests);
    public void deleteTest(Long id);
    public Test getTest(Long id);
    public List<Test> listTest();
    public Paginator listFinishedTest(int pageNumber);
    public Paginator listUnFinishedTest(int pageNumber);
    public List<String> getFreeAddresses();
    public String getFreeAddress();
}
```

Výpis 12: TestService.java

Jak můžeme vidět, všechny servisní třídy jsou definovány rozhraním. Konkrétní implementaci si ukážeme na třídě *ScenarioServiceImpl* 22.

Pro servisní třídy je typická anotace *@Service* a velice důležitá anotace *@Transactional*, pokud nám servisní metoda provádí i databázové operace. Můžeme si také povšimnout anotace *@Qualifier*, která oznamuje, o kterou konkrétní implemenentaci FileDetailsStrategy ve FileHandleru se jedná, tudíž vidíme použití návrhového vzoru strategie. Konkrétní implementaci popisuje diagram tříd 16.



Obrázek 16: Třídní diagram pro ScenarioService a InjectedCsvService

Xml konfiguraci a nastavení konkrétního argumentu konstruktoru můžeme vidět zde 13.

```
<bean id="injectedCsvFileHandler" class="com.nohaapav.sippui.file.FileHandler">
    <constructor-arg index="0" ref="injectedCsvFileDetailsStrategy"/>
</bean>

<bean id="scenarioFileHandler" class="com.nohaapav.sippui.file.FileHandler">
    <constructor-arg index="0" ref="scenarioFileDetailsStrategy"/>
</bean>

<bean id="injectedCsvFileDetailsStrategy"
  class="com.nohaapav.sippui.file.InjectedCsvFileDetailsStrategy"/>

<bean id="scenarioFileDetailsStrategy"
  class="com.nohaapav.sippui.file.ScenarioFileDetailsStrategy"/>
```

Výpis 13: spring-servlet.xml FileHandler-Strategy

6.3.2 View (Pohled)

Základní kámen pohledu tvoří technologie JSP(Java Server Pages), která používá dále tyto knihovny:

- ***Spring tags*** *(http://www.springframework.org/tags)* - především k tvorbě formulářů a lokalizací
- ***Jstl core*** *(http://java.sun.com/jsp/jstl/core)* - základní příkazy pro generováni obsahu
- ***Jstl functions*** *(http://java.sun.com/jsp/jstl/functions)* - základní operace
- ***Decorator*** *(http://www.opensymphony.com/sitemesh/decorator)* - dekorování obsahu

Dále zde najdeme kaskádové styly(CSS), které se starají o design rozhraní a také jQuery knihovnu, která zajišťuje dynamické generování obsahu. Částečně je zde použita knihovna jQueryUI, která poskytla základní vzhled našemu rozhraní.

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<table id="scenarios" class="ui-widget">
    <thead>
        <tr class="ui-widget-header ">
            <th>Name</th>
                <th>Type</th>
                <th>Created</th>
                <th>File Type</th>
                <th>Actions</th>
            </tr>
     </thead>
     <tbody>
          <c:if  test="${!empty list}">
          <c:forEach items="${list}" var="item">
                <tr>
                    <td>${item.name}</td>
                      <td>${item.type}</td>
                      <td>${item.created}</td>
                      <td>${item.fileType}</td>
                      <td>
                        <a href="delete/${item.id}">delete</a>
                        <a href="view/${item.id}">view</a>
                      </td>
                </tr>
            </c:forEach>
        </c:if>
    </tbody>
</table>
```

Výpis 14: scenario/show.jsp Scenario table

Kompletní seznam všech pohledů:

- **injectedCsv**

  - view.jsp - obsah csv souboru
  - show.jsp - list všech csv souborů příslušného uživatele
- **scenario**
  - view.jsp - obsah scénáře
  - show.jsp - list všech scénařů příslušného uživatele
- **user**
  - show.jsp - list všech uživatelů aplikace
- **test**
  - view.jsp - zobrazí testové detaily
  - list.jsp - zobrazí list všech testů
  - monitor.jsp - konzole s průběhem testu
  - form.jsp - zobrazí formulář ke spuštění testu
  - processesList.jsp - zobrazí list procesů
- login.jsp - stránka určená k přihlašování
- admin.jsp - vyrenderované, když chce uživatel načíst admin stránku, ale nemá práva
- common.jsp - domovská stránka aplikace
- denied.jsp - vyrenderované, když chce uživatel načíst stránku, ale nemá práva
- sip.jsp - stránka s tipy

Velká část kontextu, hlavně v pohledu *test/view.jsp* je generovaná pomocí jQuery, popřípadě pluginem jqPlot, který slouží na renderování grafů 15 v aplikaci.

```
$.jqplot('chart1', [data], {
        height: 200,
        grid:{
            borderColor:'transparent',
            shadow:false,
            drawBorder:false,
            shadowColor:'transparent'
        },
        seriesDefaults:{
            renderer:$.jqplot.PieRenderer,
            rendererOptions: {
                showDataLabels: true,
                dataLabels: myLabels,
                padding: 10,
                sliceMargin: 2
            }
        },
```

```
        legend: {
            show: true,
            rendererOptions: {
                numberColumns: 3
            },
            location : 'e'
        }
});
```

Výpis 15: graph.js Graf

6.3.3 Controller (Řadič)

Aplikaci řídí šest hlavních řadičů a to:

- *SippUserController*
- *LoginController*
- *ScenarioController*
- *InjectedCsvController*
- *TestController*
- *MainController*

Je patrné, co který řadič dělá, jelikož už známe funkce jednotlivých domén, a tak si rozebereme detailněji SippUserController 23. Tento řadič slouží pouze uživatelům aplikace zprávy admina, jelikož jak už bylo v analýze zmíněno, pouze uživatel v roli admina má právo registrovat a mazat uživatele aplikace.

Jak můžeme zaznamenat, na implementaci SippUserControlleru musí každý řadič obsahovat anotaci *@Controller* a bývá dobrým zvykem přidat i mapováni za pomocí anotace *@RequestMapping*. Každá další metoda řadiče obsahuje své vlastní mapování a typ požadavku (RequestMethod). SippUserController obsahuje také anotaci *@PathVariable* v metodě list, která nám dává možnost posílat do metody řadiče parametr v jeho cestě. V metodě save si opět můžeme všimnout anotace *@ModelAttribute*, která představuje objekt formuláře určeného k zpracování. Za returnem je rovněž možnost všimnout si klíčového slova *redirect*, které odkazuje na jinou metodu řadiče. Pokud slovo redirect chybí, řadič odkáže na příslušný pohled, nebo v případě anotace *@ResponseBody* vrátí příslušná data.

6.3.4 Component (Komponenty)

V této sekci si ukážeme implementaci jednoduché testové komponenty, kterou představuje vlákno TestWorker 16 a ExecutorThreadPoolHandler 17, která nám slouží jako thread pool.

```
@Override
    public SippProcess call() {
        showCommand();
```

```
        Integer fpid = null;
        String pid = null;
        ProcessBuilder processBuilder = new ProcessBuilder(commands);
        try {
            final Process process = processBuilder.start();
            fpid = getUnixPID(process);
            try (BufferedReader stdInput = new BufferedReader(new InputStreamReader(process.
                 getInputStream()))) {
                String inputStr;
                while ((inputStr = stdInput.readLine()) != null) {
                    if (inputStr.contains("PID")) {
                        pid = inputStr.substring(inputStr.indexOf("[") + 1, inputStr.indexOf("]"))
                            ;
                    }
                }
            }
        } catch (IOException | IllegalArgumentException | IllegalAccessException |
             NoSuchFieldException ex) {
            log.error(ex);
        }

        return new SippProcess(pid, fpid, type);
    }
```

Výpis 16: TestWorker.java Funkce call()

TestWorker nejdřív vytvoří z formu který byl konstruktorem předán uac nebo uas příkaz a pak proces spustí, přičemž vrátí PID procesu, aby bylo možné kdykoli daný proces ukončit respektive s ním provádět jiné operace. Jednotlivé procesy jsou vytvářeny pomocí ExecutorThreadPoolHandleru, který se spolu s komponentou ProcessWorker stará o to, aby zátěž byla rozdělena konstantně mezi jádra procesoru. Po skončení procesů je zavolaná metoda *cleanUp*, která vypne danou komponentu.

```
@Component
public class ExecutorThreadPoolHandler {

    private ExecutorService executor;

    public ExecutorService getExecutor() {
        return executor;
    }

    public void init () throws Exception {
        executor = Executors.newCachedThreadPool();
    }

    public void cleanUp() throws Exception {
        executor.shutdownNow();
    }
}
```

Výpis 17: ExecutorThreadPoolHandler.java

Hlavní testové komponenty systému jsou:

- *ProcessWorker* - spouští testové procesy a procesy na generováni ip adres
- *IpWorker*- zastupuje jednoduchý proces na generováni ip adresy
- *TestWorker* - zastupuje jednoduchý testový proces
- *ExecutorThreadPoolHandler* - spouštení (metoda *executeSippBatch 18*) a správa jednotlivých procesů

```
public void executeSippBatch(final TestForm form) {
    final List<TestWorker> list = initializeForm(form);
    Test test = testService.createTest(TType.UAC);

    try {
        List<Future<SippProcess>> results = executor.getExecutor().invokeAll(list);
        for (Future<SippProcess> result : results) {
            final SippProcess process = result.get();
            if (process.getProcessType().equals(TType.UAS)){
                test = testService.createTest(TType.UAS);
            }
            log.debug("Process pid: " + process.getPid());
            log.debug("Process fpid: " + process.getFpid());
            process.setTest(test);
            processService.createProcess(process);
        }
    } catch (ExecutionException | InterruptedException ex) {
        log.error(ex);
    }
}
```

Výpis 18: Funkce executeSippBatch()

Hlavní komponenty na práci se soubory jsou:

- *FileHandler* - všeobecné operace na práci ze soubory
- *CountFileHandler*- parser a filtr pro data z count souborů generovaných testerem
- *ErrorFileHandler* - parser a filtr pro data z chybových souborů generovaných testerem

6.4 Konfigurace

Jelikož je rozhraní postaveno na frameworku Spring, umožnuje velice snadnou konfiguraci jak už databázových prvků, tak i jiných atributů, a to díky objektu PropertyPlaceholderConfigurer 19, který je nakonfigurován v souboru *spring-servlet.xml*.

```
<bean id="propertyConfigurer"
        class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
        p:location="/WEB-INF/app.properties" />

<bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"
```

```
p:driverClassName="${jdbc.driverClassName}"
p:url="${jdbc.databaseurl}"
p:username="${jdbc.username}"
p:password="${jdbc.password}" />
```

Výpis 19: Nastavení komponenty propertyConfigurer

Jak můžeme vidět, propertyConfigurer 19 nastavuje jako externí konfigurační soubor *app.properties* 20 tudíž všechna data, která je potřeba konfigurovat, a která se mohou často měnit, je potřeba konfigurovat v tomto souboru. Boubor obsahuje prozatím jen konfiguraci databáze, počet jader procesoru serveru, na kterém beží a cestu ke složce, kde si server ukladá potřebna xml data uploadována uživatelem.

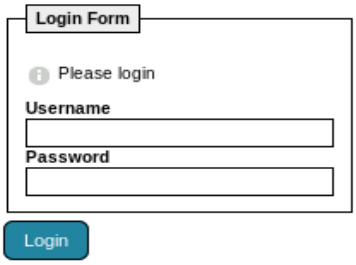
```
# ORACLE DB
jdbc.driverClassName= oracle.jdbc.driver.OracleDriver
jdbc.dialect=org.hibernate.dialect.Oracle10gDialect
jdbc.databaseurl=jdbc:oracle:thin:@localhost:1521:XE
jdbc.username=dp_dev
jdbc.password=user7917

# APP CORE
thread.count=10

# APP STARTUP
application.root.path=/home/nohaapav/Apps/sippUI/
```

Výpis 20: Soubor app.properties

7 Dokumentace

7.1 Autentizace

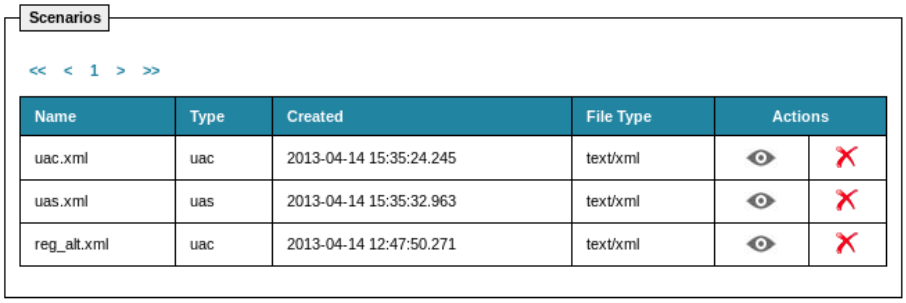
Po zadání příslušné url do webového prohlížeče je uživatel vyzván k zadání loginu a hesla. Po úspešné autentizaci je odkázan na hlavní domovskou stránku aplikace, kde si může z menu vybrat, zda-li chce spouštět nové testy nebo také navštívit jiné sekce, jako například sekci se scénaři. Pokud uživatel z nějakého důvodu autentizací neprojde, je informován o neúspěchu v záhlaví logovacího formuláře 17.



Obrázek 17: Logovací formulář

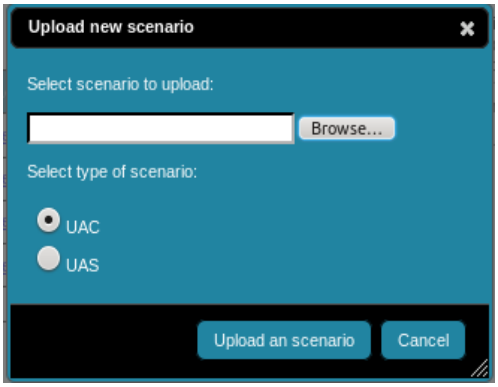
7.2 Správa scénařů

Sekci scénařu uživatel najde v menu *Tests* a po kliknutí na položku *Scenario* je uživateli zobrazena přehledná tabulka 18 se všemi scénaři uploadovanými daným uživatelem.



Obrázek 18: Tabulka scénařů

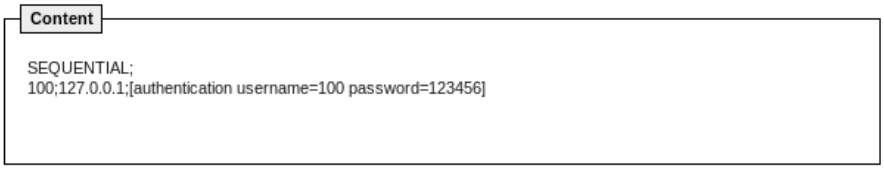
Jak si můžeme povšimnout, uživatel si může jednotlivé scénaře prohlížet za pomocí akce *view*, nebo je mazat *delete*. Na stránce se nachází pod tabulkou se scénáři také tlačítko *Upload Scenario*, které uživateli otevře nové vyskakovací okno 19 a umožní nahrát nový scénář, buďto typu uac nebo uas.



Obrázek 19: Nahráni nového scénaře

7.3 Správa csv filů

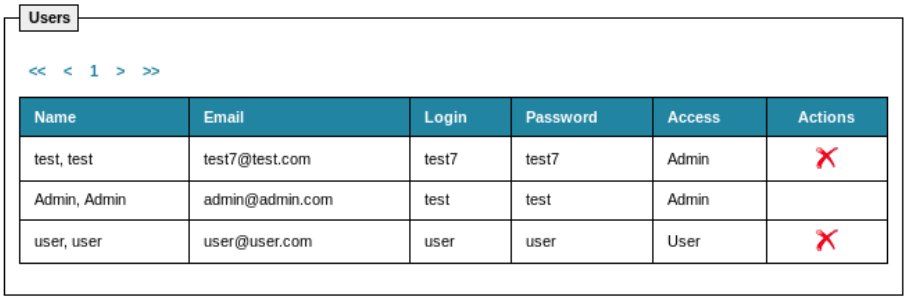
Správa externích souborů funguje víceméně na stejné bázi jako scénáře. Po zvolení sekce *InjectedCsv* z menu *Tests* je uživateli zobrazena tabulka se všemi csv soubory, které nahrál. Ke každé položce listu jsou taktéž dostupné akce *delete* pro mazání a *view* pro detailní pohled na obsah souboru 20. Nechybí ani tlačítko *Upload csv file* pro otevření vyskakovacího okna, které slouží k nahrání daného csv souboru na server.



Obrázek 20: Obsah csv souboru

7.4 Správa uživatelů

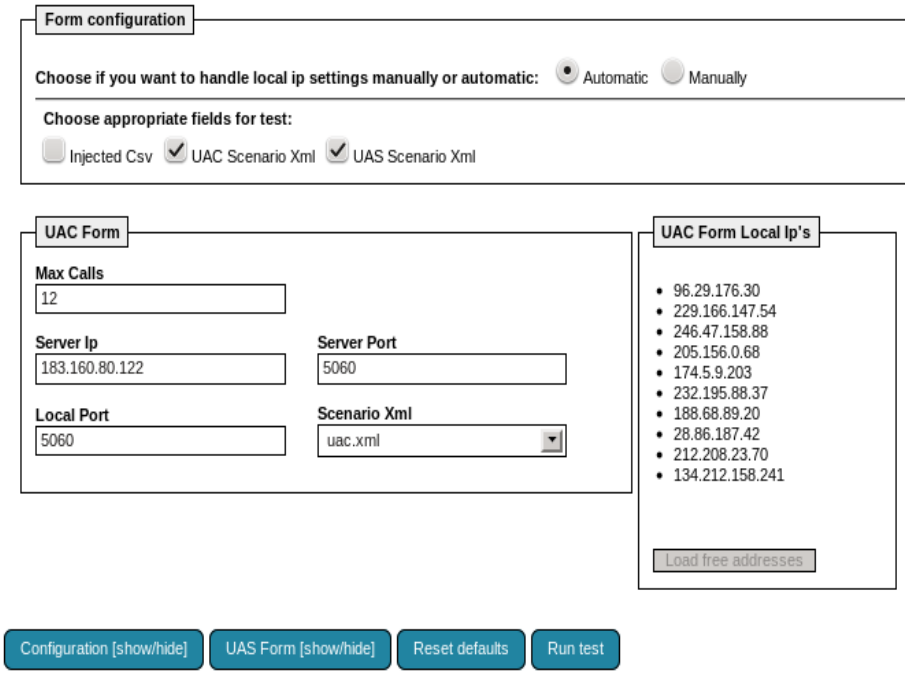
Uživatelská sekce je dostupná jen pro uživatele v roli admina a slouží k registraci nových uživatelů aplikace. Najdeme ji v menu *Admin*, v položce *Users*. Po načtení sekce má admin možnost vidět seznam všech uživatelů, 21 aplikace s jejich rolemi a samozřejmě detailními informacemi. Jak můžeme vidět, uživatel má právo mazat ostatní uživatele pomocí akce *delete*, ale nemůže smazat sám sebe. Rovněž může registrovat nové uživatele, a to za pomoci tlačítka *Create User*, které se nachází pod seznamem a otvírá registrační formulář ve formě vyskakovacího okna. Pokud jsou údaje zvalidovány a je uživateli přiřazena příslušná role, může začít pracovat se systémem.



Obrázek 21: Seznam uživatelů

7.5 Vytvoření testu

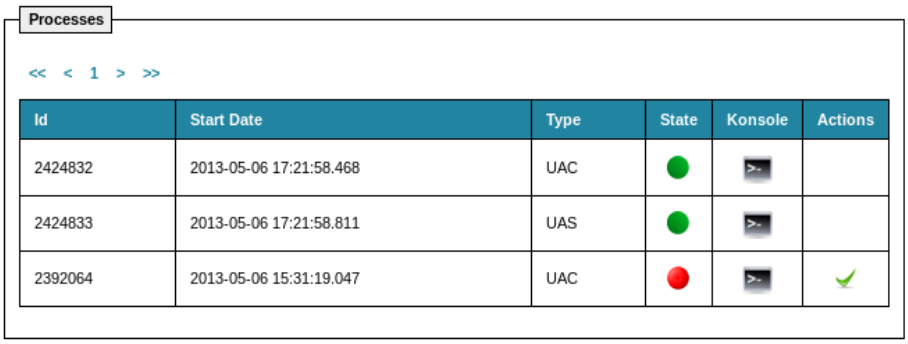
Testy uživatel vytváří v záložce *Test form*. Tato sekce obsahuje tlačítko *Configuration [show/hide]*, které otevře sekci s konfigurací testu, jenž umožní uživateli navolit si potřebné paremetry testu a také si vybrat mezi automatickým či manuálním generováním IP adres. Při jednosměrných testech uživatel vyplňuje pouze Uac formulář, kdežto při obousměrných jak Uac, tak Uas formulář 22. Na zobrazení a aktivaci Uac formuláře slouží tlačítko *Uac Form [show/hide]*. Po vyplnění všeho potřebného a následné validaci je spuštěn testový proces.



Obrázek 22: Testový formulář

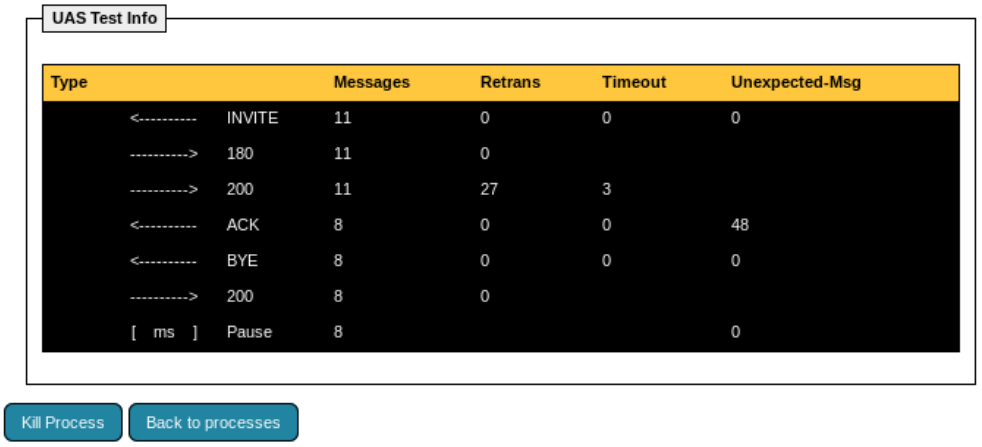
7.6 Správa testových procesů

Po spuštění testu má uživatel možnost sledovat průběh testu v záložce *Test processes* v menu *Tests*. Je zde monitor 23, který obsahuje seznam testů a jejich momentální stav. Uživatel má možnost otevřít si konzoli 24 k danému testu, kde může sledovat průběh, nebo naopak proces ukončit.



Obrázek 23: List procesů

Po ukončení testového procesu je uživateli zobrazená ikona která slouží k uložení testových výsledku do databáze, kde je užival vyzván k zadání názvu testu a volitelného popisku. Po úspěšném uložení je test přesunut do tabulky testů a testový proces smazán.



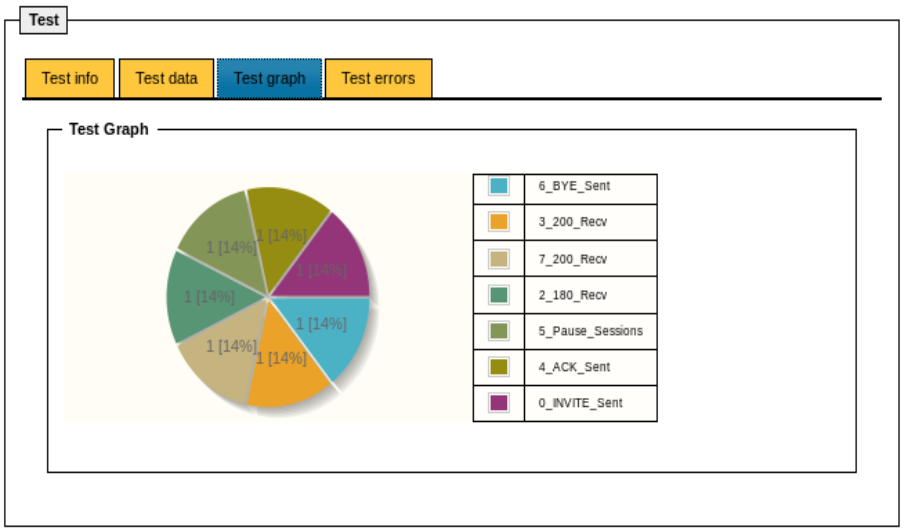
Obrázek 24: Konzole s průběhem testu

7.7 Správa testů

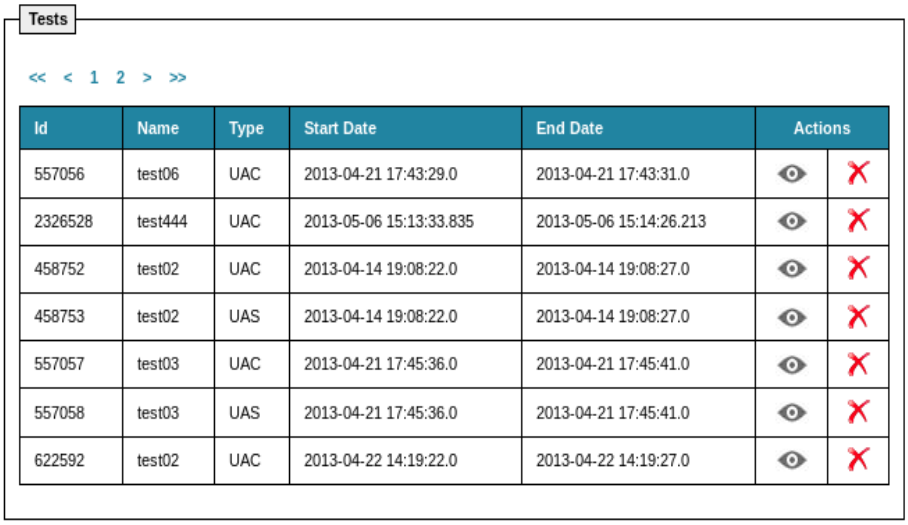
V sekci *Test Results* v menu *Tests*, si uživatel může výsledky testů prohlížet, nebo je mazat 27. Akce *view* zobrazí celkový přehled o průběhu testu. Obsahuje čtyři záložky a to:

- Test info - základní informace o testu

- Test data - dílčí data testu
- Test graph - graf znázorňující četnost SIP zpráv 25
- Test errors - chyby, které nastaly při testu

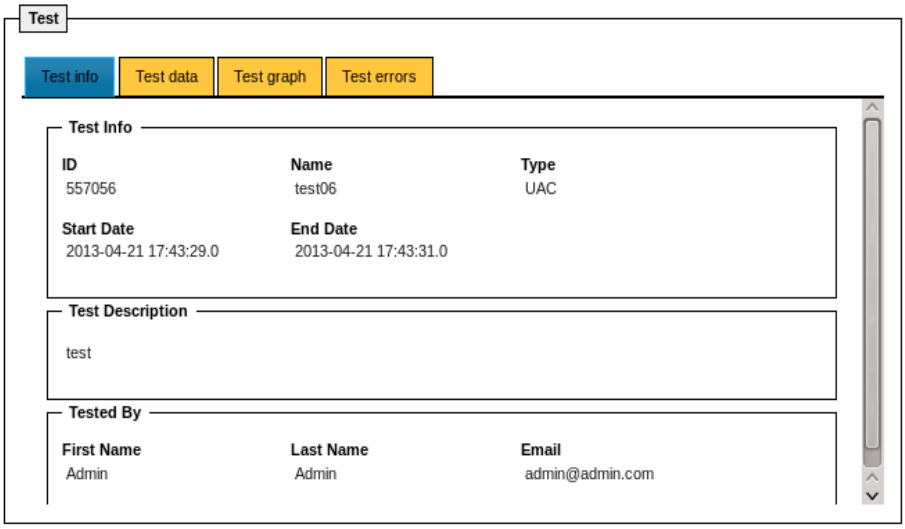


Obrázek 25: Graf četnosti sip správ



Id	Name	Type	Start Date	End Date	Actions
557056	test06	UAC	2013-04-21 17:43:29.0	2013-04-21 17:43:31.0	
2326528	test444	UAC	2013-05-06 15:13:33.835	2013-05-06 15:14:26.213	
458752	test02	UAC	2013-04-14 19:08:22.0	2013-04-14 19:08:27.0	
458753	test02	UAS	2013-04-14 19:08:22.0	2013-04-14 19:08:27.0	
557057	test03	UAC	2013-04-21 17:45:36.0	2013-04-21 17:45:41.0	
557058	test03	UAS	2013-04-21 17:45:36.0	2013-04-21 17:45:41.0	
622592	test02	UAC	2013-04-22 14:19:22.0	2013-04-22 14:19:27.0	

Obrázek 26: Seznam testů



Obrázek 27: Detail testu

8 Závěr

Úkolem webového rozhraní pro správu procesů výkonnostního testování SIP infrastruktury je zefektivnit testové procesy vhodným přerozdělením mezi jednotlivá jádra procesoru a taktéž oprostit uživatele od současného, velice nepohodlného procesu testování. Aplikace nahrazuje zdlouhavé a mnohdy komplikované CLI příkazy přehledným formulářem a dává uživateli možnost pohodlně si navolit vše potřebné ke spuštení testového procesu. Velký důraz je také kladen na validaci dat před spuštením testů, čímž se aplikace snaží uživatele nasměrovat k logickému formování SIP příkazů. Uživatel zajisté také uvítá přechod od csv souborů k databázi co se týče práce s daty. Díky faktu, že se jedná o webové rozhraní, je uživatel oprošten od různých konfigurací a má možnost se k aplikaci připojit téměř odkudkoli.

V první části práce se snažím přiblížit uživateli SIP protokol a jeho testovací nástroj SIPp. Můžeme zde najít různé příklady SIP správ a také jednoduchých SIPp testů. Další část práce se věnuje otázce vývoje rozhraní. Najdeme zde klady a zápory spojené s implementací rozhraní. Následuje kapitola, která popisuje technologie použité při vývoji rozhraní z informativního hlediska a také odpověď na otázku, proč jsem si danou technologii k implementaci zvolil. Dostáváme se k části návrhové, která popisuje analýzu a návrh aplikace ze softwarového hlediska a nalezneme v ní různé případy užití a databázové návrhy. Implementační sekce naproti tomu dovolí nahlédnout do zdrojových kódu, uml diagramů a různých konfigurací aplikace. Poslední částí práce je jednoduchá uživatelská dokumentace lemována obrázky z ui pro snadnou orientaci, která popisuje všechny případy užití aplikace.

Rozhraní pro správu procesů výkonnostního testování SIP infrastruktury využívá množství knihoven a nástrojů, které splňují definici otevřeného softwaru. Proto je také rozhraní nositelem statusu otevřeného softwaru, kterým bude zajištěn další rozvoj a zužitkováno úsilí, které bylo investováno do vývoje.

Pavol Noha

9 Reference

[1] SINNREICH, H. *SIP Beyond VoIP,* VON Publishing LLC, New York, 2005.

[2] VOZNAK, M. *Voice over IP,* VSB-Technical University of Ostrava: College Textbook, 1st. ed., Ostrava, 2008.

[3] VOZNAK M., SAFARIK J. *DoS Attacks Targeting SIP Server and Improvements of Robustness,* ISSUE 1, VOLUME 6, 2012.

[4] VOZNAK M. *SIP versus H.323,* str 9, HiPath Spektrum, 02/2005, čtvrtletně vydává AC&C Public Realations, Praha, 2005.

[5] VOZNAK M., ROZHON J. *SIP Back to Back User Benchmarking,* Proceedings The Sixth International Conference on Wireless and Mobile Communications, pp. 92-96, Published by IEEE Computer Society, University of Valencia, Spain, September 2010.

[6] VOZNAK M., ROZHON J. *Approach to stress tests in SIP environment based on marginal analysis,* In Journal SPRINGER: Telecommunication Systems, 11p., June 2011.

[7] VOZNAK M., REZAC F., TOMALA K. *SIP Penetration Test System,* Proceedings The 34th Conference on Telecommunications and Signal Processing, pp. 504-508, Vienna, Austria, August 2010.

[8] Webové stránka *SIPp*, http://sipp.sourceforge.net/doc/reference.html#Foreword

[9] Webové stránka *Cesnet*, https://sip.cesnet.cz/cs/protokoly/sip

[10] Webové stránka *Programujte*, http://programujte.com/clanek/2007040702-java-tutorial-technologie-1-dil/

[11] Webové stránka *Spring*, http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/overview.html

[12] Webové stránka *Oracle*, http://docs.oracle.com/javaee/5/tutorial/doc/bnbqa.html

[13] Webové stránka *Apache*, http://openjpa.apache.org/builds/1.0.2/apache-openjpa-1.0.2/docs/manual/jpa_overview_em_lifecycle.html

[14] Webové stránka *Muni*, http://kore.fi.muni.cz/wiki/index.php/Maven

A Přílohy

A.1 Zdrojové kódy aplikace

Součásti práce jsou zdrojové kódy webové aplikace pro správu procesů výkonnostního testování SIP infrastruktury. Aplikace používá Maven, což znamená, že všechny závislosti jsou automaticky stáhnuty při buildu aplikace z daných repozitářů. Pokud by build selhal kvůli chybějící závislosti, stačí patřičný balík dohledat a přidat repozitář.

V aplikaci je jeden konfigurační soubor, který je potřeba před startem aplikace nastavit.

src/main/webapp/WEB-INF/app.properties - potřeba nastavit počet jader procesoru, cestu ke složce, kde budou ukládána uživatelská data a také přístup k databázi.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http:// www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
             maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>cz.vsb.noh031</groupId>
    <artifactId>sipp</artifactId>
    <version>1.0</version>
    <name>Evidence Scenaru</name>

    <developers>
        <developer>
            <name>Pavol Noha</name>
            <email>noh031@vsb.cz</email>
            <url>http://www.vsb.cz/peoples/noh031</url>
        </developer>
    </developers>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    </properties>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>7</source>
                    <target>7</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</ artifactId >
            <version>4.10</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

Výpis 21: Příklad minimální konfigurace .pom souboru

```
@Service
public class ScenarioServiceImpl implements ScenarioService {

    private static final Logger log = Logger.getLogger("service");
    @Autowired
    @Qualifier("scenarioFileHandler")
    private FileHandler fileHandler;
    @Autowired
    private ScenarioDAO scenarioDAO;
    private static final int PAGE_SIZE = 8;

    @Override
    @Transactional
    public void uploadScenario(final MultipartFile file, final String type) {
      //imlementation comes here
    }

    @Override
    @Transactional
    public Paginator listScenario(final int pageNumber) {
        final Portfolio portfolio = Session.getUserDetails().getSippUser().getPortfolio();
        final List<Scenario> list = scenarioDAO.listScenario(portfolio);
        return new Paginator(list, pageNumber, PAGE_SIZE);
    }

    @Override
    @Transactional
    public List<Scenario> listScenario() {
        final Portfolio portfolio = Session.getUserDetails().getSippUser().getPortfolio();
        return scenarioDAO.listScenario(portfolio);
    }

    @Override
    @Transactional(readOnly = true)
    public Scenario getScenario(final Long id) {
        return scenarioDAO.findScenarioById(id);
    }

    @Override
    @Transactional
    public void deleteScenario(final Long id) {
        final Scenario scenario = scenarioDAO.findScenarioById(id);
        final boolean deleted = fileHandler.deleteFile(scenario.getName());
        if (deleted) {
            scenarioDAO.deleteScenario(scenario);
        }
    }

}
```

Výpis 22: ScenarioServiceImpl.java

```
@Controller
@RequestMapping("/user")
public class SippUserController {

    private static final Logger log = Logger.getLogger("controller");
    @Autowired
    private UserService userService;
    @Autowired
    private PortfolioService portfolioService;

    @RequestMapping(value = "/show/{pageNumber}", method = RequestMethod.GET)
    public String listUser(@PathVariable int pageNumber, ModelMap map) {
        Paginator page = userService.listUser(pageNumber);

        List<SippUser> data = (List<SippUser>) page.getData();
        int current = page.getPageNumber();
        int begin = Math.max(1, current - 5);
        int end = Math.min(begin + 10, page.getTotalPages());

        map.put("list", data);
        map.put("beginIndex", begin);
        map.put("endIndex", end);
        map.put("currentIndex", current);
        map.put("totalPages", page.getTotalPages());
        map.put("user", new SippUser());

        return "user/show";
    }

    @RequestMapping(value = "/show/save", method = RequestMethod.POST)
    public String save(@ModelAttribute("user") SippUser user, BindingResult result) {
        if (user != null) {
            userService.createUser(user);
            userService.createUserFolder(user.getLogin());
            Portfolio userPortfolio = new Portfolio();
            userPortfolio.setUser(user);
            portfolioService.createPortfolio(userPortfolio);
        }
        return "redirect:/user/show/1";
    }

    @RequestMapping(value = "/show/delete/{userId}", method = RequestMethod.GET)
    public String deleteUser(@PathVariable("userId") Long userId) {
        SippUser user = userService.findUserById(userId);
        if (user != null) {
            userService.deleteUser(user);
            userService.deleteUserFolder(user.getLogin());
        }
        return "redirect:/user/show/1";
    }
}
```

Výpis 23: SippUserController.java